

ARTICLE

MVFN-FG: LLM-Driven Deobfuscation and Multi-View Fusion Network for Fine-Grained WebShell Detection

Di Yao¹, Yuling Chen^{1,*}, Xiuzhang Yang^{1,*}, Xuwei Wang², Haiwei Sang³, Weijie Tan^{1,4} and Zhi Ouyang¹

¹Guizhou Provincial Laboratory of Big Data, State Key Laboratory of Public Big Data, Guizhou University, Guiyang, China

²Computer College, Weifang University of Science and Technology, Weifang, China

³Guizhou Key Laboratory of Artificial Intelligence and Brain-inspired Computing, College of Mathematics and Big Data, Guizhou Education University, Guiyang, China

⁴Guizhou Provincial Laboratory of Big Data, College of Computer Science and Technology, Guizhou University, Guiyang, China

*Corresponding Authors: Yuling Chen. Email: ylchen3@gzu.edu.cn; Xiuzhang Yang. Email: xzyang@gzu.edu.cn

Received: 04 June 2026; Accepted: 26 August 2026; Published: 22 October 2026

ABSTRACT: In recent years, WebShell attacks have exploited web application scripting languages to enable persistent remote control, unauthorized command execution, and flexible post-exploitation operations, making them a widely adopted technique in phishing campaigns, advanced persistent threat (APT) operations, and Internet of Things (IoT) network environments. Existing WebShell detection methods mainly focus on benign-malicious binary classification, while fine-grained malicious behavior analysis remains insufficient. Moreover, heterogeneous scripting languages, code obfuscation, syntactic mutation, and dynamic function invocation further limit the robustness of traditional lexical, syntactic, or structural detection approaches. To address these challenges, we propose MVFN-FG, a novel WebShell detection framework for robust malicious code detection and fine-grained behavior classification. MVFN-FG first employs a large language model to perform semantic-aware deobfuscation and recover the hidden execution logic of obfuscated WebShell scripts. The reconstructed code is then used to construct four complementary feature views, namely WordShell, TokenShell, ASTShell, and FlowShell, which capture lexical patterns, token-level semantics, syntactic structures, and execution-flow dependencies. An adaptive multi-view fusion mechanism is designed to integrate heterogeneous representations, and a hybrid ResNet-BiLSTM-Transformer model is developed to learn hierarchical semantic representations and global behavioral dependencies. Experimental results demonstrate that MVFN-FG improves robustness against complex code mutations and accurately supports both malicious WebShell detection and fine-grained behavior classification. The proposed framework provides effective technical support for WebShell detection and malicious script analysis.

KEYWORDS: WebShell; multi-view fusion; fine-grained behavior classification; ResNet-BiLSTM-Transformer

1 Introduction

With the pervasive integration of the Internet of Things into modern infrastructures, IoT systems have become indispensable components of contemporary society, while their large-scale interconnectivity significantly expands the cyber-attack surface. Among various intrusion vectors, WebShell attacks leveraging malicious scripts to maintain persistent unauthorized access have emerged as a prominent threat to IoT systems security [1]. In IoT systems, heterogeneous components and distributed management interfaces collectively increase exposure to adversarial exploitation. These characteristics create opportunities for

attackers to bypass authentication mechanisms and deploy WebShells for remote command execution (RCE), data exfiltration, and lateral movement across interconnected IoT systems. According to Zscaler, IoT-targeted malware campaigns have increased by more than 400% year-on-year [2], with WebShells increasingly serving as an initial intrusion vector in advanced persistent threat (APT) campaigns targeting IoT systems.

At present, WebShell detection primarily relies on rule-based matching and static feature libraries, such as ModSecurity and YARA rules [3]. While these methods are engineering-efficient, they heavily depend on manual rule updates and exhibit limited adaptability against advanced script obfuscation, syntax compression, and cross-language variants, often resulting in high false positive rates [4]. In IoT systems, WebShells further enhance stealthiness through multi-language implementations (PHP, ASP, JSP) and Abstract Syntax Tree (AST)-level rewriting techniques, which introduce semantic drift and significantly increase detection difficulty [5]. In addition, IoT systems are typically resource-constrained and heterogeneous, making it difficult to deploy heavy dynamic analysis or sandbox-based inspection mechanisms. This creates a strong demand for detection frameworks that jointly consider semantic understanding and lightweight feature extraction [6]. Fig. 1 illustrates the lifecycle of a WebShell attack within a managed IoT system environment, showing how a single stealthy entry point escalates into a full-scale compromise.

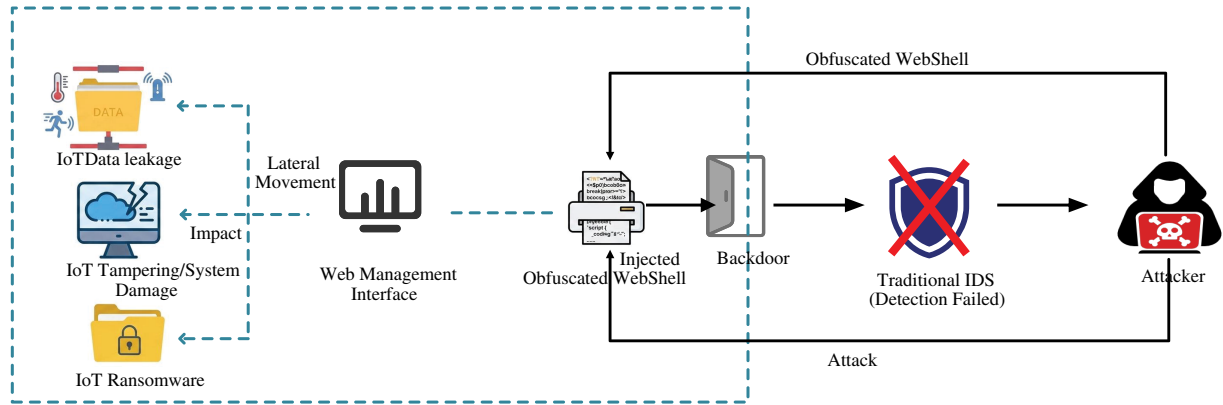


Figure 1: IoT security threats caused by WebShell.

As deep learning and natural language processing techniques continue to advance, machine learning-based frameworks such as SVM, RF, and CNN have been widely adopted to automate feature extraction for malicious code analysis [7]. Conventional approaches leveraging multi-view fusion strategies [8] or opcode sequence modeling [9] have demonstrated effectiveness on large-scale datasets. More recently, embedding-based frameworks such as PowerDetector [10], PowerPeeler [11], and Power-ASTNN [12] have further improved obfuscated script detection through dynamic de-obfuscation mechanisms and AST-based neural architectures. However, existing approaches still face several practical limitations in IoT systems: first, limited robustness against multi-language and multi-structure hybrid scripts due to reliance on single-representation paradigms; second, insufficient capability in handling heavily obfuscated samples involving dynamic injection or multi-layer encoding; and third, lack of fine-grained behavioral understanding, as most existing methods remain constrained to coarse-grained binary classification tasks.

To overcome these gaps, we propose MVFN-FG, a fine-grained WebShell detection framework that integrates LLM-driven deobfuscation, multi-view feature representation, adaptive fusion, and hierarchical classification. Its novelty lies in building a semantic-to-behavior pipeline for obfuscated WebShells, where LLM-based reconstruction is combined with WordShell, TokenShell, ASTShell, and FlowShell features

to capture lexical, token-level, syntactic, and execution-flow behaviors. Unlike existing methods that mainly rely on static signatures, single-view representations, or coarse-grained binary detection, MVFN-FG provides a more comprehensive representation for detecting obfuscated WebShells and recognizing fine-grained WebShell behaviors.

In summary, the main contributions of this paper are as follows:

- **LLM-driven semantic deobfuscation for WebShell analysis:** We design an LLM-driven semantic reconstruction module to recover the hidden execution logic of obfuscated WebShell scripts. Unlike conventional lexical restoration or rule-based deobfuscation methods, the proposed module aims to restore encoded strings, dynamic function calls, and concealed execution paths while preserving the original malicious semantics.
- **Multi-view behavioral representation for fine-grained WebShell detection:** We construct four complementary feature views, namely WordShell, TokenShell, ASTShell, and FlowShell, to jointly capture lexical patterns, token-level semantics, syntactic structures, and execution-flow dependencies. This design enables MVFN-FG to move beyond benign-malicious binary detection and support fine-grained classification of WebShell malicious behaviors.
- **Adaptive semantic fusion and hierarchical classification:** We propose an adaptive multi-view fusion mechanism to dynamically integrate heterogeneous feature representations, and combine it with a hybrid ResNet-BiLSTM-Transformer model to learn local syntax patterns, sequential dependencies, and global semantic relationships. Compared with simple feature concatenation, the proposed fusion strategy better captures cross-view dependencies among WebShell behaviors.
- **Comprehensive evaluation against learning-based methods, security tools, and model variants:** Extensive experiments on real-world multi-language WebShell datasets demonstrate that MVFN-FG outperforms traditional machine learning models, deep learning baselines, representative prior methods, mature security tools, and internal model variants in both malicious WebShell detection and fine-grained behavior classification. The results verify the effectiveness of the proposed semantic reconstruction and multi-view fusion design.

2 Related Work

This section reviews the malicious threats in the Internet of Things system and summarizes the existing WebShell detection methods, classifying them according to their underlying technologies.

2.1 Malicious Threats in IoT

IoT environments face diverse security challenges, including ransomware, botnets, zero-day exploits, phishing, and stealthy command-and-control (C2) communications [13,14]. To mitigate these malware threats [15,16], researchers have proposed defenses spanning representation learning [17], network traffic detection [18], variant identification [19], propagation modeling [20], and edge autonomy [21]. For script-based and WebShell threats, effective pipelines require integrating multi-language analysis, deobfuscation, and runtime monitoring to counter multi-stage dynamic evasion [22,23].

2.2 WebShell Detection Based on Rule and Structural Analysis

Early detection paradigms rely heavily on static rule configuration, template matching, and expert knowledge to identify malicious behaviors [24–26]. While highly interpretable, traditional rule-based frameworks struggle with advanced obfuscation and cross-language generalization. To address these gaps, semantic ranking algorithms and deep classifiers have been introduced to model malicious intents and

analyze instructions structurally [27,28]. However, maintaining these rules in resource-constrained IoT nodes remains challenging due to the polymorphic nature of modern lightweight WebShells.

2.3 WebShell Detection Based on Machine Learning and Deep Learning

To automate feature engineering, machine learning and deep learning approaches have been widely adopted [29]. Machine learning models utilize lexical features, access logs, and ensemble frameworks to achieve rapid static identification [30–32]. Concurrently, deep learning architectures—including CNNs for spatial request context modeling [33–36], MLP and L-matrix optimizations for deployment efficiency [37,38], and TreeLSTM/RNNs for abstract syntax tree (AST) opcode classification [39,40]—have improved abstraction and obfuscation resilience. Nevertheless, these methods heavily depend on high-quality training features or intact syntax structures, limiting their real-time edge applicability.

2.4 WebShell Detection via Behavioral Graphs and Semantic Enhancement

Recent advancements leverage Graph Neural Networks (GNNs), AST-heterogeneous graphs, and word-level semantic dependencies to capture structural call relationships and behavioral paths [41,42]. Despite enhancing structural alignment, existing methods exhibit limitations in cross-language flexibility, fine-grained behavior recognition, and handling sophisticated encryption, as summarized in Table 1.

Table 1: Related work comparison.

Related Work	Techniques	Focus	FE-Dep	MLing	Obfus	SIA	SemM	Robus
Fang et al., 2018	RF + FastText	Rule-based	✓	×	×	×	×	×
Pan et al., 2021	PHP-feature ext.	Behavior ext.	✓	×	○	×	○	○
Lee et al., 2024	TextRank	Obfus-semantic	○	✓	○	×	✓	✓
Wu et al., 2019	Session-ML	Log-behavior	○	✓	✓	×	○	✓
Yong et al., 2022	Ensemble-learning	IoT-env	○	✓	✓	×	○	✓
Yang et al., 2018	CNN + Word2Vec	HTTP-request	×	✓	✓	×	✓	✓
Tian et al., 2017	CNN	Static-class.	×	×	✓	×	○	✓
Rose et al., 2025	AST + DL	Malicious-sample	×	✓	✓	✓	✓	✓
Shang et al., 2024	TreeLSTM + AST	Multi-lang AST	×	✓	✓	✓	✓	✓
Hannousse et al., 2023	GNN + semantic	Semantic-graph	×	✓	✓	✓	✓	✓
MVFN-FG	AST + LLM + Hybrid	Multi-dim collab	×	✓	✓	✓	✓	✓

Note: FE-Dep: Feature-engineering reliance. MLing: Multilingual support. Obfus: Obfuscation adaptability. SIA: Structural-integration ability. SemM: Semantic modeling. Robus: Robustness. ✓ indicates full coverage or strong capability in the corresponding aspect. ○ denotes partial coverage or limited capability. × means no coverage or lack of capability. The stronger the obfuscation adaptability, the more accurately the approach can identify advanced WebShell attack samples involving encryption, morphing, or behavior concealment.

In summary, existing approaches either emphasize static rule analysis or focus on structural model design, but they all exhibit significant limitations, as they only support benign-malicious binary classification and lack fine-grained malicious behavior recognition. To address these issues, this paper proposes a multi-source detection framework that integrates abstract syntax tree parsing and LLM-based linguistic deobfuscation. This framework enables joint modeling of multi-perspective features and semantic-driven identification, offering a new pathway for accurate WebShell detection in complex malicious code analysis tasks.

3 Methodology

[Section 3](#) introduces the overall framework of MVFN-FG, followed by a detailed overview of the implementation processes for each stage and its corresponding algorithms.

3.1 Overview

To cope with the increasingly complex WebShell attacks and variants of backdoors, this study innovatively designs and implements MVFN-FG, a novel fine-grained WebShell behavior detection framework based on LLM-driven deobfuscation and multi-view semantic fusion. The overall architecture is illustrated in [Fig. 2](#) and consists of four stages, which are described as follows:

- (1) **WebShell Code Extraction and Deobfuscation:** At this initial stage, the system scans WebShell-related benign and malicious script files, such as PHP, ASP, and JSP files. Specifically, accurate WebShell instances are identified and extracted based on a diverse set of predefined rules, providing a reliable data foundation for subsequent analysis. To counter sophisticated adversarial techniques such as control-flow distortion and multi-layer encoding, we leverage a Large Language Model (e.g., GPT-4o) with specialized prompt engineering. This process resolves semantic ambiguities and reconstructs the execution logic, transforming obfuscated scripts into semantically equivalent, simplified code.
- (2) **AST Parsing and Structural Modeling:** To capture the underlying logic of the deobfuscated scripts, the framework utilizes language-specific parsers (e.g., PHP-Parser, JavaParser, JScriptParser) to generate Abstract Syntax Trees (ASTs). This structural representation maps the source code into a hierarchical tree format, facilitating the extraction of robust structural and semantic features that are invariant to simple syntax mutations.
- (3) **Multi-dimensional Feature Extraction and Fusion:** In this stage, we extract heterogeneous features across four dimensions: WordShell, TokenShell, FlowShell, and ASTShell. These features capture a comprehensive range of characteristics, from lexical distribution to control-flow dependencies. The extracted features are subsequently vectorized into high-dimensional dense embeddings and fused into a unified feature space, providing a rich informational foundation for behavioral analysis.
- (4) **Model Building and Fine-grained Classification:** The fused vectors are fed into a hybrid architecture composed of ResNet, BiLSTM, and Transformer for direct benign-malicious classification and fine-grained malicious behavior classification. The model maps the extracted semantic representations directly to specific attack categories, including command execution, file manipulation, and privilege escalation. Detailed comparative experiments verify the effectiveness and robustness of MVFN-FG.

3.2 WebShell Code Extraction and LLM-Based WebShell Deobfuscation

As shown in [Fig. 3](#), WebShell scripts are processed through a three-stage preprocessing pipeline. First, file filtering is performed based on extensions and header signatures (e.g., .php, .asp, .jsp) to remove non-script artifacts. Second, keyword and pattern-based analysis is applied to detect suspicious API calls (e.g., eval, system, exec) and common obfuscation patterns (e.g., base64_decode), while behavioral similarity is measured using the Longest Common Subsequence (LCS). Third, normalization and cleaning are applied to remove redundant whitespace, comments, and syntactic noise. A code slicing strategy is further introduced to extract execution-relevant segments while preserving control-flow dependencies.

To mitigate data leakage risks, a lightweight sanitization process is applied before invoking the LLM. Sensitive runtime entities, including IP addresses, domain names, URLs, file paths, and email addresses, are replaced with standardized placeholders (<IP>, <DOMAIN>, <URL>, <PATH>, <EMAIL>), preserving structural semantics while eliminating exposure of operational information.

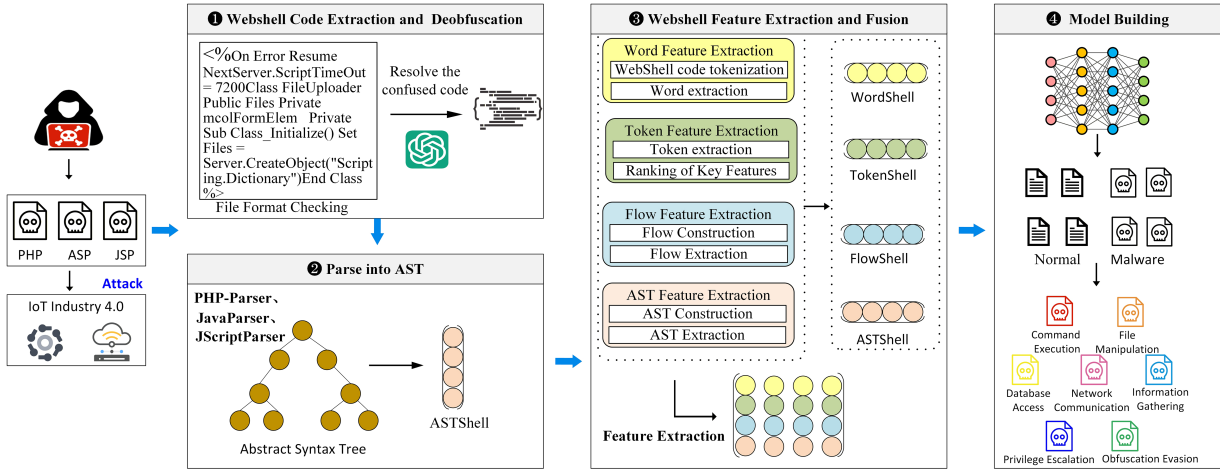


Figure 2: The architecture of MVFN-FG.

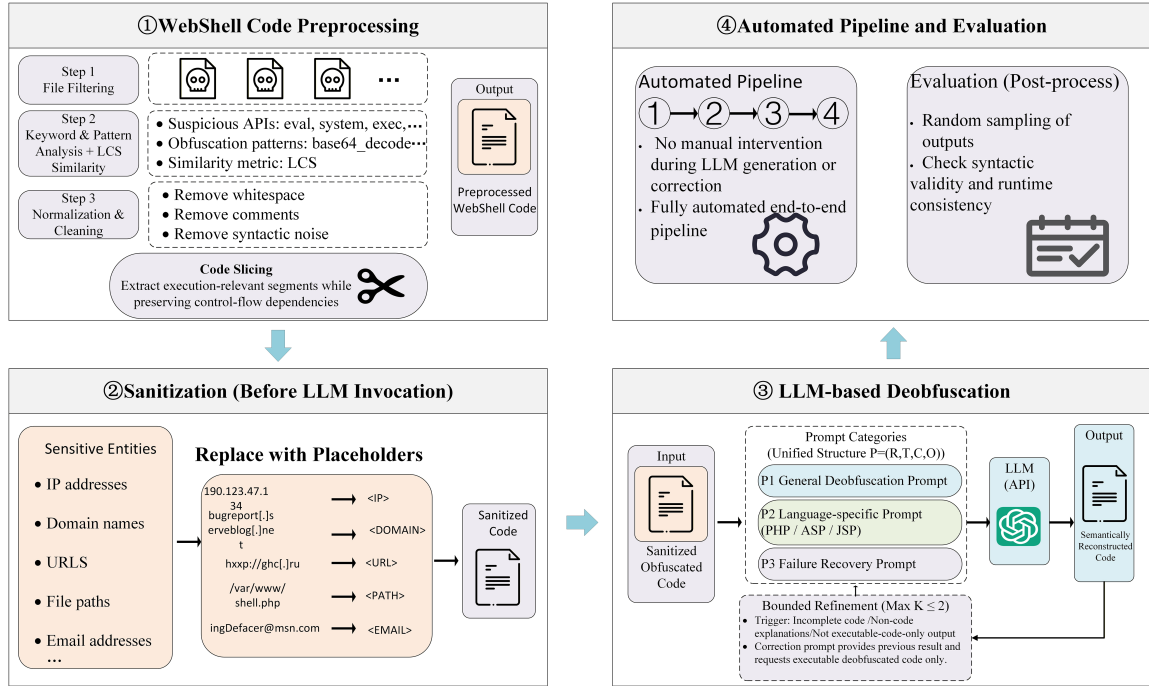


Figure 3: Pipeline of LLM-based preprocessing and semantic reconstruction for WebShell.

The GPT-4o-based deobfuscation module is not part of the real-time, line-rate IDS processing path. Instead, it is used as an auxiliary semantic-analysis component for suspicious WebShell scripts extracted during preprocessing. Raw live network traffic is not directly transmitted to the external GPT-4o API. Only extracted WebShell code or execution-relevant code slices are submitted to GPT-4o via the API after the sanitization procedure described above. Therefore, the external LLM is used to reconstruct the semantics of selected suspicious scripts rather than to perform real-time packet-level traffic inspection. The de-obfuscation module is formally described as follows:

$$\mathcal{F}_{LLM} : X_{obf} \rightarrow X_{clean}, \quad (1)$$

where X_{obf} denotes obfuscated WebShell code and X_{clean} denotes semantically reconstructed code.

We construct three prompt categories: general deobfuscation prompt P_1 , language-specific prompt P_2 (PHP/ASP/JSP), and failure recovery prompt P_3 . All prompts follow a unified structure $P = (R, T, C, O)$, where R denotes role definition, T task instruction, C constraints, and O output format specification. The prompts enforce execution logic preservation, decoding of obfuscated strings, variable renaming, and restoration of dynamic function calls, while restricting output to executable code.

For prompt implementation, we design three prompt templates corresponding to the three prompt categories, namely the general deobfuscation template, the language-specific deobfuscation template, and the failure recovery template. For each obfuscated WebShell sample, the general deobfuscation template is first used. If the script language can be identified as PHP, ASP, or JSP, the corresponding language-specific template is applied to provide syntax-aware constraints. When the generated result contains incomplete code, non-code explanations, or does not satisfy the executable-code-only output requirement, the failure recovery template is triggered for bounded correction. Thus, each sample is processed with one initial deobfuscation prompt and at most two additional correction prompts.

A representative prompt template is defined as follows:

Role: You are a code security analyst specializing in WebShell deobfuscation.

Task: Decode the following obfuscated WebShell code and reconstruct its original execution logic.

Constraints: Preserve the original semantics, restore encoded strings and dynamic function calls, rename unclear variables when necessary, and do not introduce new functionality.

Output: Return only executable deobfuscated code without additional explanations.

Due to space limitations, only a representative prompt is presented in this section, while the complete unified prompt template is provided in [Appendix A](#). The LLM inference process adopts a single-pass generation strategy with bounded refinement. When the generated output is incomplete or fails to satisfy the required output format, a maximum of $K \leq 2$ correction iterations is triggered. The correction prompt provides the previous generation result and explicitly requests executable deobfuscated code without additional explanations. No manual intervention is introduced during the LLM generation or correction process. Human inspection is only used in the evaluation stage to verify a randomly sampled subset of deobfuscated outputs in terms of syntactic validity and runtime consistency. All inputs are sanitized before being fed into the model to prevent sensitive operational data from being leaked.

3.3 WebShell Abstract Syntax Tree Generation

Following preprocessing and multi-stage screening, we implement a structural analysis based on the Abstract Syntax Tree (AST) to map the syntax structures and contextual dependencies of WebShells. To generate consistent ASTs across heterogeneous languages, we utilize PhpParser for PHP, JavaParser for JSP, and JScriptParser for ASP. The AST module extracts multi-granularity features, including node-level attributes (e.g., sensitive function calls, variable flow, nesting depth) and global structural properties (e.g., maximum nesting depth, sensitive API distribution, suspicious input ratio). This AST-based approach transforms raw source code into hierarchical semantic trees, enabling subsequent deep learning frameworks to capture hidden malicious behaviors through comprehensive syntax and context understanding.

3.4 WebShell Feature Extraction

To extract holistic behavioral representations across multiple dimensions, we implement four complementary feature extraction methods. First, we integrate word, token, and AST structures into three semantic vectors. WordShellVec employs code semantic normalization to filter high-frequency malicious terms (e.g., *eval*, *assert*, *\$_GET*, *\$_POST*) and maps them into a continuous embedding space using a Skip-Gram framework. TokenShellVec tokenizes the deobfuscated code into basic syntax units $T = \{t_1, t_2, \dots, t_n\}$, including keywords, operators, and identifiers, which are converted into a dense feature matrix to adapt to structural mutations. To counter control-flow obfuscation, ASTShellVec parses scripts into multi-branch tree structures via language-specific parsers and extracts critical semantic paths through level-by-level traversal, such as $\text{FunctionDef} \rightarrow \text{Expr} \rightarrow \text{Eval}$, to vectorize structural syntax characteristics.

Subsequently, FlowShellVec Feature Extraction models attack intent through execution logic by constructing a global execution flow view that combines Control Flow Graphs (CFG) and Data Flow Graphs (DFG). Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT) are employed to extract structural dependency features:

$$\mathbf{v}^{cfg} = \text{GCN}(G_{CFG}), \quad \mathbf{v}^{dfg} = \text{GAT}(G_{DFG}) \quad (2)$$

where G_{CFG} and G_{DFG} denote the control and data flow graph structures, respectively. The ultimate flow-level representation is derived via vector concatenation as $\mathbf{V}^{(flow)} = [\mathbf{v}^{cfg} \parallel \mathbf{v}^{dfg}]$, which captures execution branching structures and multi-stage command causal logic to detect complex, distributed attack workflows.

3.5 Model Construction

As illustrated in Fig. 4, we propose MVFN-FG, a hybrid neural network architecture that formulates WebShell detection as a hierarchical representation learning and fine-grained classification task. The framework is logically structured into two primary stages: (1) spatiotemporal feature extraction and fusion via a combined ResNet and BiLSTM module, and (2) global semantic modeling and multi-granularity behavior classification based on a Transformer encoder-decoder structure. This design enables the model to progressively learn from heterogeneous feature spaces to high-level semantic behaviors, facilitating the precise identification of specific malicious activities.

Adaptive Attention-based Multi-View Fusion. To effectively integrate heterogeneous features, we propose an adaptive attention-based fusion mechanism that dynamically learns the contribution of each feature view. Unlike fixed-weight fusion, this approach enables the model to emphasize informative views based on semantic relevance, which is crucial for handling obfuscated WebShell behaviors. Mathematically, the attention-based multi-view fusion process diagrammed in Algorithm 1 is formulated as:

$$\mathbf{V}^{(multi)} = \sigma \left(\sum_i \alpha_i \cdot (\mathbf{W}_p^{(i)} \mathbf{V}^{(i)} + \mathbf{b}_p^{(i)}) \right) \quad (3)$$

where $\mathbf{W}_p^{(i)}$ and $\mathbf{b}_p^{(i)}$ denote the projection parameters for the i -th view. The dynamic attention weight α_i is computed via a softmax function over the alignment scores: $\alpha_i = \exp(\mathbf{s}_i) / \sum_j \exp(\mathbf{s}_j)$, where the score is defined as $\mathbf{s}_i = \mathbf{q}^\top \tanh(\mathbf{W}_a \mathbf{V}_{align}^{(i)})$, and $\mathbf{q}^\top$, $\mathbf{W}_a$ represent the attention network weights.

Hybrid ResNet-BiLSTM-Transformer Architecture. The combination of ResNet, BiLSTM, and Transformer leverages their complementary modeling capabilities to extract a hierarchical representation spanning local syntax patterns to global semantic behaviors. Specifically, ResNet captures local invariant patterns and mitigates gradient degradation via identity shortcuts. BiLSTM then models bidirectional

sequential dependencies to capture execution logic and contextual relationships within WebShell scripts. Finally, the Transformer network enhances global dependency modeling through self-attention, enabling the detection of long-range interaction patterns and complex attack chains.

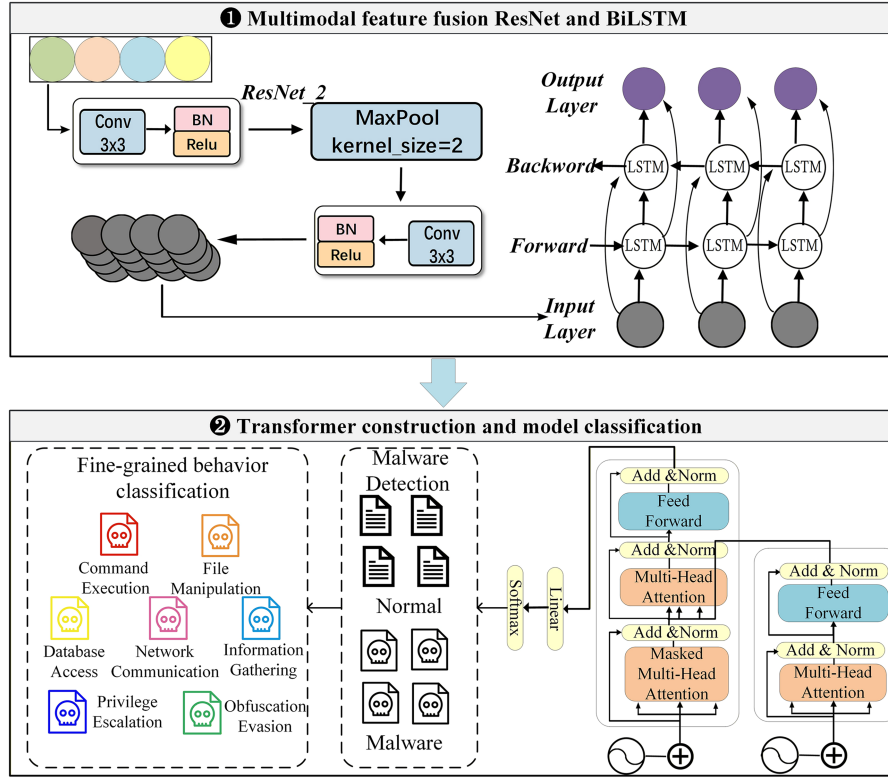


Figure 4: Adaptive attention-based multi-view fusion architecture for WebShell detection.

The fused multi-view representation $\mathbf{V}^{(multi)} \in \mathbb{R}^d$ is sequentially propagated through this hierarchical backbone to derive the global context representation $\mathbf{Z}_L$. First, the local features are extracted via the ResNet block:

$$\mathbf{H}_{res} = \text{ReLU} \left(\text{BN} \left(\text{Conv1D}(\mathbf{V}^{(multi)}) + \mathbf{V}^{(multi)} \right) \right) \quad (4)$$

Then, the global behavioral dependencies are modeled through the cascaded BiLSTM and Transformer layers:

$$\mathbf{Z}_L = \text{Transformer}^L (\text{BiLSTM}(\mathbf{H}_{res})) \quad (5)$$

Optimization and Classification. The final prediction probability vector is obtained via a standard dense layer with softmax activation, expressed as $\hat{y} = \text{Softmax}(\mathbf{W}_c \mathbf{Z}_L + \mathbf{b}_c)$, where $\mathbf{W}_c$ and $\mathbf{b}_c$ are the classification parameters. To optimize the fine-grained behavior classification, we adopt a standard multi-class cross-entropy loss function defined as:

$$\mathcal{L} = - \sum_k y_k \log(\hat{y}_k) \quad (6)$$

where y_k and $\hat{y}_k$ denote the true and predicted probabilities for the k -th behavior category, respectively. Compared with traditional fixed-weight fusion, this fine-grained optimization combined with adaptive attention selection significantly elevates the model's robustness against complex obfuscation and cross-language variations in resource-constrained IoT environments.

Algorithm 1: Adaptive attention-based multi-view feature fusion

Input: Heterogeneous feature matrices $\mathbf{V} = \{\mathbf{V}^{(t)}, \mathbf{V}^{(a)}, \mathbf{V}^{(w)}, \mathbf{V}^{(f)}\}$
Output: Fused multi-view representation $\mathbf{V}^{(multi)}$

```

1 for each feature view  $\mathbf{V}^{(i)} \in \mathbf{V}$  do
2    $\mathbf{V}_{align}^{(i)} \leftarrow \text{ProjectToCommonSpace}(\mathbf{V}^{(i)});$ 
3    $\mathbf{s}_i \leftarrow \text{ComputeAttentionScore}(\mathbf{V}_{align}^{(i)});$ 
4 end
5  $\alpha \leftarrow \text{Softmax}(\mathbf{s});$  //Compute dynamic alignment weights
6  $\mathbf{V}^{(multi)} \leftarrow \text{WeightedPooling}(\alpha, \mathbf{V}_{align});$ 
7 return  $\mathbf{V}^{(multi)}$ 

```

4 Experiments

This section implements the MVFN-FG and conducts detailed comparative experiments to highlight the effectiveness and robustness of the proposed detector over the baseline methods. We evaluate MVFN-FG through the following research questions:

- RQ1:** Is MVFN-FG more efficient than other detection tools in handling WebShell code? Can it accurately distinguish between malicious and benign code as well as different types of malicious behaviors? (See [Section 4.2](#))
- RQ2:** How does the multi-angle fusion technology contribute to the overall performance of MVFN-FG? (See [Section 4.3](#))
- RQ3:** Can MVFN-FG capture deep-seated semantic relationships? (See [Section 4.4](#))
- RQ4:** Does the proposed large model de-obfuscation algorithm outperform existing methods and successfully recover various WebShell malicious readable source code? (See [Section 4.5](#))

4.1 Dataset and Experimental Environment

For evaluation, as shown in [Table 2](#), we collected a total of 29,248 script files written in PHP, JSP, and ASP, comprising 21,665 benign scripts (White) and 7583 malicious WebShell scripts (Black). These samples were collected from GitHub repositories such as LandGrey (2020), JohnTroony (2021), Hziec (2022), and Tanjiti (2022). The dataset is stratified split into training and testing sets with a 7:3 ratio. Experiments were conducted in a Windows 11 environment with an AMD Ryzen 9 9955HX CPU, NVIDIA GeForce RTX 5060 Laptop GPU, and 16 GB memory, implemented via Python (3.10) and PyTorch. To demonstrate the advantages of MVFN-FG, we establish several control groups for comparison: traditional machine learning models, such as RF; classical deep learning architectures, such as CNN-BiLSTM; state-of-the-art baseline frameworks, such as TextRank; and two model variants, namely ResNet-BiLSTM-Transformer (Obfuscated) and ResNet-BiLSTM-Transformer (Concat). The ResNet-BiLSTM-Transformer (Concat) model adopts a simple feature concatenation strategy without explicit interaction modeling, which limits its ability to capture feature dependencies. The ResNet-BiLSTM-Transformer (Obfuscated) model is directly evaluated on obfuscated samples before de-obfuscation processing. To ensure fairness, all models adopt the same activation function (ReLU), normalization method, and dropout rate (0.1).

Table 2: Lists the detailed information of these datasets.

Sources	Type	Number	Dataset Class
LandGrey	PHP, JSP, ASP	18	Malicious Sample
JohnTroony	PHP	119	Malicious Sample
HzieeC	PHP, JSP, ASP	26,738	Normal Sample and Malicious Sample
Tanjiti	PHP, JSP, ASP	506	Malicious Sample
China-Eugene	PHP, JSP, ASP	1867	Malicious Sample
Total	PHP, JSP, ASP	29,248	Normal Sample and Malicious Sample

Model performance is evaluated using accuracy, recall, precision, and F1-score based on True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). We report multiple-run variance, estimate 95% confidence intervals using bootstrap sampling, and assess statistical significance using paired *t*-tests. A *p*-value lower than 0.05 is considered to indicate a statistically significant difference. In binary WebShell detection, TP refers to correctly detected WebShell files, TN refers to correctly identified normal files, FP refers to normal files incorrectly classified as WebShell, FN refers to WebShell files that are not detected, and support denotes the number of samples in each category. In fine-grained multi-class behavior classification (e.g., `command_execution`), TP refers to correctly predicted samples of the target class, TN refers to correctly predicted samples of all other classes, FP refers to samples from other classes incorrectly classified as the target class, FN refers to target-class samples incorrectly classified into other classes, and support denotes the number of samples per class.

4.2 Malicious WebShell Code Detection and Fine-Grained Behavioral Analysis (RQ1)

To evaluate the effectiveness of MVFN-FG in binary benign-vs-malicious WebShell detection, we conduct binary classification experiments in which each input script is classified as either benign or malicious. We compare the proposed model with the control groups mentioned in [Section 4.1](#), as well as existing mature WebShell detection tools, including YARA rules, ClamAV, ModSecurity, and AI-SHIELD. The corresponding binary detection results are reported in [Table 3](#).

As shown in [Table 3](#), MVFN-FG achieves the best overall performance in the binary detection task, with an Accuracy of 0.9898, an F1-score of 0.9855, a Precision of 0.9799, and a Recall of 0.9915. Compared with Random Forest, which achieves an F1-score of 0.8490, and K-Nearest Neighbors, which achieves an F1-score of 0.8316, MVFN-FG improves the F1-score by 13.65% and 15.39%, respectively. Compared with deep learning baselines, including CNN-BiLSTM and CNN-BiLSTM-Transformer, MVFN-FG further achieves improvements of 12.82% and 2.89% in F1-score. For representative methods from prior literature, MVFN-FG consistently outperforms TextRank (0.9016), DNN (0.9732), and TF-IDF-based methods (0.9802) in terms of F1-score, demonstrating its effectiveness in distinguishing malicious WebShell scripts from benign scripts. Among traditional security tools, YARA rules achieve an Accuracy of 0.8069 and an F1-score of 0.5741, with a limited Recall of 0.5020, indicating weak generalization to unseen or obfuscated WebShell samples. ClamAV obtains an Accuracy of 0.7991 and a Precision of 0.9058, but its Recall is only 0.2512, resulting in a low F1-score of 0.3934. This suggests that ClamAV can accurately identify some known WebShell signatures but misses many variant or obfuscated samples. ModSecurity achieves a higher Accuracy of 0.9610 and an F1-score of 0.9248, but it remains inferior to AI-SHIELD and MVFN-FG, showing that rule-based HTTP inspection is less effective than learning-based semantic detection for complex WebShell samples. ResNet-BiLSTM-Transformer (Obfuscated) achieves an F1-score of 0.8931, while MVFN-FG improves the F1-score by 9.24%. ResNet-BiLSTM-Transformer (Concat) achieves an F1-score of 0.9316, whereas MVFN-FG further

improves it by 5.39%. Overall, the results indicate that MVFN-FG outperforms both learning-based and rule-based baseline methods in binary benign-vs-malicious WebShell detection. The statistical results further support the reliability of the binary detection experiment. To evaluate training-run variability, MVFN-FG was independently retrained five times using random seeds 42, 43, 44, 45, and 46, while the stratified training and test partitions were kept fixed across all runs. The mean F1-score across the five independent runs was 0.9855, and the sample variance of the five run-level F1-scores was 1.4×10^{-6} . Therefore, the reported multiple-run variance reflects variability across independently retrained MVFN-FG models using different random seeds. For MVFN-FG, the 95% confidence interval of the F1-score is [0.9839, 0.9870]. Compared with the main baseline methods, all paired significance tests produced p -values lower than 0.05, indicating that the performance improvement of MVFN-FG in binary WebShell detection was statistically significant.

Table 3: Performance comparison of different models for binary benign-vs-malicious WebShell detection.

Model	Accuracy	F1-Score	Precision	Recall
KNN	0.9005	0.8316	0.9428	0.7836
LR	0.9027	0.8375	0.9441	0.7951
RF	0.9105	0.8490	0.9183	0.8026
SVM	0.8870	0.8241	0.8578	0.8009
CNN	0.9188	0.8799	0.8919	0.8694
BiLSTM	0.9035	0.8403	0.9064	0.7999
CNN + BiLSTM	0.9105	0.8573	0.9096	0.8245
CNN + BiLSTM + Transformer	0.9693	0.9566	0.9701	0.9639
TextRank [27]	0.9233	0.9016	0.9123	0.8924
DNN [29]	0.9782	0.9732	0.9662	0.9809
Tfidf [7]	0.9841	0.9802	0.9765	0.9841
YARA rules	0.8069	0.5741	0.6704	0.5020
ClamAV	0.7991	0.3934	0.9058	0.2512
ModSecurity	0.9610	0.9248	0.9247	0.9248
AI-SHIELD	0.9790	0.9699	0.9790	0.9689
ResNet-BiLSTM-Transformer (Obfuscated)	0.9042	0.8931	0.9093	0.8775
ResNet-BiLSTM-Transformer (Concat)	0.9539	0.9316	0.9468	0.9183
MVFN-FG	0.9898	0.9855	0.9799	0.9915

After evaluating binary detection, we further investigate a distinct multi-class fine-grained behavior classification task. In this task, only malicious WebShell samples are considered, and each sample is classified into one of seven fine-grained behavior categories: command execution, file manipulation, database access, information gathering, network communication, privilege escalation, and obfuscation evasion. The corresponding multi-class classification results are presented in Fig. 5.

The comparison includes traditional machine learning models, deep learning architectures, and representative feature engineering-based methods. Since traditional security tools such as ClamAV and ModSecurity do not support fine-grained behavior classification, they are not included in the comparison for the multi-class task. As illustrated in Fig. 5, each algorithm is evaluated using a multi-view representation for multi-class fine-grained WebShell behavior classification. MVFN-FG achieves the highest F1-score of 0.9923, outperforming all the baseline models. Compared with traditional machine learning models, MVFN-FG improves the F1-score by 7.61% over KNN, 15.84% over Logistic Regression, 6.14% over Random

Forest, and 8.15% over SVM. Compared with deep learning baselines, MVFN-FG improves the F1-score by 6.82% over CNN, 8.05% over BiLSTM, 1.39% over CNN-BiLSTM, and 0.55% over CNN-BiLSTM-Transformer. For representative feature engineering-based and prior methods, MVFN-FG also achieves better performance than TextRank [27], DNN [29], and Tfidf [7], with F1-score improvements of 3.79%, 10.05%, and 3.19%, respectively. This indicates that although these methods can capture lexical or statistical patterns to some extent, they remain limited in modeling complex semantic dependencies and fine-grained behavioral correlations. Furthermore, MVFN-FG improves the F1-score by 7.23% over the un-deobfuscated ResNet-BiLSTM-Transformer variant and by 5.34% over the simple feature concatenation baseline. Overall, the results in Fig. 5 confirm the effectiveness of MVFN-FG in learning robust multi-view representations for multi-class fine-grained WebShell behavior classification. The statistical results also confirm the stability of the fine-grained behavior classification task. Following the same repeated-training protocol used in the binary detection experiment, MVFN-FG achieved a mean F1-score of 0.9923 across five independent runs. The sample variance of the five run-level F1-scores was 8.6×10^{-7} , reflecting variability across independently retrained models. For MVFN-FG, the 95% confidence interval of the F1-score is [0.9907, 0.9938]. The paired significance tests against the main multi-class baselines yielded p -values lower than 0.05, further demonstrating the statistically significant advantage of MVFN-FG. Table 4 reports the class-wise performance of MVFN-FG. The results for database_access, network_communication, and obfuscation_evasion should be interpreted with caution because their limited test support makes the metrics sensitive to individual prediction outcomes. These values should therefore be regarded as point estimates from the current test set.

4.3 Comparison of Single-View and Multi-View Methods for Fine-Grained WebShell Behavior Detection (RQ2)

To evaluate the impact of single-view and multi-view feature representations on fine-grained WebShell behavior detection, Table 5 and Fig. 6 compare the performance of different models under WordShell, TokenShell, ASTShell, FlowShell, and unified multi-view settings. Overall, most models achieve their best or near-best performance under the multi-view configuration, indicating that different feature views provide complementary information for behavior characterization. As shown in Fig. 6, the red star markers representing the multi-view results are generally located on the right side of the single-view markers in both the Accuracy and F1-score plots, visually confirming the advantage of multi-view fusion. In particular, MVFN-FG is positioned at the far right in both subfigures, demonstrating its leading performance among all compared methods.

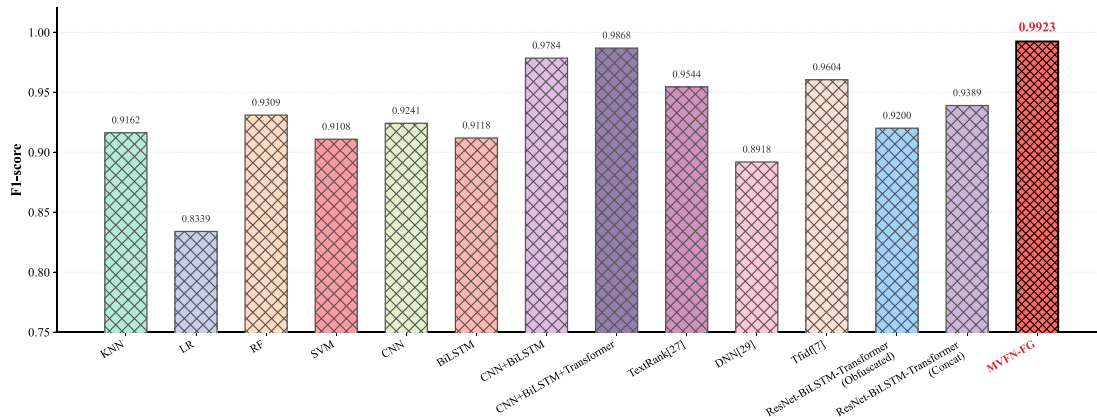


Figure 5: Performance comparison of different models for multi-class fine-grained WebShell behavior classification.

Table 4: Performance and class-wise sample sizes of multi-class malware behavior classification.

Malware Behavior	Precision	Recall	F1	Train Set	Test Set
Command_execution	1.0000	0.9709	0.9852	1134	486
File_manipulation	1.0000	1.0000	1.0000	3332	1428
Database_access	0.9987	1.0000	0.9993	72	31
Information_gathering	0.9506	1.0000	0.9747	464	199
Network_communication	0.9755	1.0000	0.9876	56	24
Privilege_escalation	0.9050	1.0000	0.9501	140	60
Obfuscation_evasion	0.9999	0.9190	0.9577	110	47
Weighted_avg/total	0.9929	0.9921	0.9923	5308	2275

Note: The per-class precision, recall, and F1-score values are point estimates obtained from the current test set. The results for categories with limited test support, particularly database_access ($n = 31$), network_communication ($n = 24$), and obfuscation_evasion ($n = 47$), may be sensitive to individual prediction outcomes and should therefore be interpreted with caution.

Table 5: Detailed performance comparison of single-view and multi-view feature representations across different classification models.

Model	WordShell		TokenShell		ASTShell		FlowShell		Multi-View	
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score
KNN	0.8289	0.8254	0.8162	0.8114	0.8119	0.8051	0.7310	0.7170	0.9174	0.9162
LR	0.7652	0.7953	0.7348	0.7718	0.7363	0.7407	0.6444	0.7049	0.8185	0.8339
RF	0.9081	0.9047	0.8940	0.8883	0.8558	0.8409	0.8382	0.8333	0.9327	0.9309
SVM	0.8584	0.8596	0.8553	0.8572	0.8545	0.8580	0.8119	0.8126	0.9103	0.9108
CNN	0.9163	0.9037	0.9186	0.9041	0.9005	0.8843	0.8977	0.8870	0.9316	0.9241
BiLSTM	0.9011	0.8845	0.9118	0.9141	0.9005	0.9040	0.8949	0.8944	0.9265	0.9118
CNN + BiLSTM	0.9553	0.9511	0.9582	0.9497	0.9542	0.9471	0.9146	0.9130	0.9791	0.9784
CNN + BiLSTM + Transformer	0.9644	0.9637	0.9808	0.9810	0.9587	0.9575	0.9124	0.9104	0.9870	0.9868
TextRank [27]	0.9233	0.9016	0.9365	0.9258	0.9412	0.9346	0.8876	0.8724	0.9570	0.9544
DNN [29]	0.8415	0.8506	0.8568	0.8624	0.8652	0.8708	0.8016	0.8125	0.8836	0.8918
Tfidf [7]	0.9286	0.9214	0.9432	0.9368	0.9475	0.9412	0.9018	0.8957	0.9600	0.9604
ResNet-BiLSTM-Transformer (Obfuscated)	0.9042	0.8931	0.9086	0.9024	0.8963	0.8897	0.8652	0.8586	0.9122	0.9200
ResNet-BiLSTM-Transformer (Concat)	0.9539	0.9316	0.9584	0.9345	0.9512	0.9283	0.9146	0.9058	0.9633	0.9389
MVFN-FG	0.9729	0.9733	0.9904	0.9905	0.9893	0.9893	0.9163	0.9153	0.9921	0.9923

For MVFN-FG, the multi-view representation achieves the highest F1-score of 0.9923. Compared with FlowShell, WordShell, ASTShell, and TokenShell, the multi-view representation improves the F1-score by 7.7%, 1.9%, 0.3%, and 0.18%, respectively. This demonstrates that the proposed fusion strategy effectively compensates for the semantic limitations of individual feature views. Fig. 6 further shows that FlowShell usually yields relatively lower performance than lexical, token-level, and AST-based views, while its integration still contributes complementary execution-flow information when combined with other views.

In addition to conventional machine learning and deep learning baselines, representative feature-based methods, including TextRank [27], DNN [29], and TF-IDF [7], also benefit from multi-view representation. Their multi-view F1-scores reach 0.9544, 0.8918, and 0.9604, respectively, which are higher than their corresponding single-view results. Moreover, the ResNet-BiLSTM-Transformer (Obfuscated) variant obtains a multi-view F1-score of 0.9200, showing that directly extracting features from obfuscated scripts still limits behavior recognition. The ResNet-BiLSTM-Transformer (Concat) variant improves the F1-score to 0.9389 through simple feature concatenation, but it remains lower than MVFN-FG, indicating that explicit adaptive

fusion is more effective than direct concatenation. These results confirm that MVFN-FG achieves more fine-grained detection of malicious WebShell behaviors by jointly modeling WordShell, TokenShell, ASTShell, and FlowShell features.

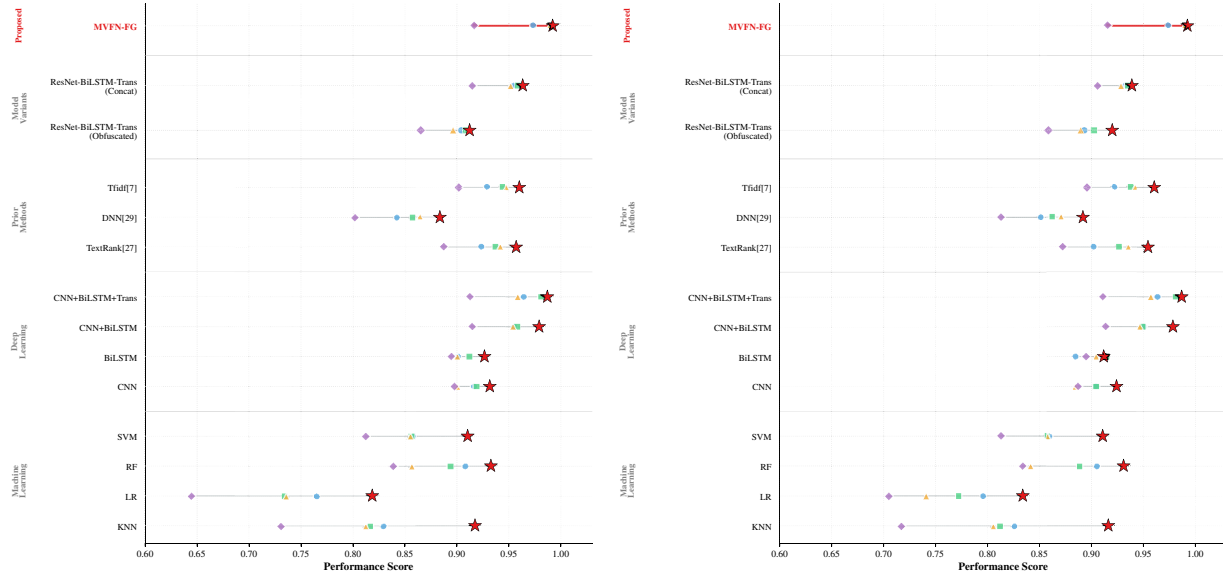


Figure 6: Accuracy and F1-score comparison illustrating the superiority of the multi-view approach over individual single-views.

Note: ● denotes WordShell, ■ denotes TokenShell, ▲ denotes ASTShell, ◆ denotes FlowShell, ★ denotes the Multi-View results.

4.4 Performance of MVFN-FG (RQ3)

To evaluate MVFN-FG's capability in semantic feature extraction and behavior clustering, we constructed a behavior-feature relationship graph for WebShell malicious code (see Fig. 7). The graph clusters key functions and tokens into seven distinct behavior clusters: `command_execution`, `file_manipulation`, `database_access`, `information_gathering`, `network_communication`, `privilege_escalation`, and `obfuscation_evasion`. The cluster distribution intuitively reflects the degree of feature concentration and interconnection under different attack intentions.

In the graph, nodes represent key tokens or functions extracted from WebShell source code, such as 'eval', 'shell_exec', 'fopen', 'mysqli_query', 'curl_init', 'base64_decode', 'chmod', and '\$_POST'. Node size reflects token frequency or classification importance, while edge thickness indicates co-occurrence frequency and association strength within the same sample. The cluster structure reveals different malicious behavior patterns. In the 'command_execution' cluster, sensitive functions frequently co-occur with input entry points such as '\$_POST' and '\$_REQUEST', indicating that external inputs are commonly used to trigger command execution. The 'file_manipulation' cluster contains functions such as 'fopen', 'fwrite', and 'unlink', reflecting file creation, modification, and deletion behaviors. The 'database_access' and 'network_communication' clusters correspond to database interaction, data access, remote download, and C2 communication behaviors. In addition, the 'obfuscation_evasion' cluster is connected with multiple behavior clusters, suggesting that encoding and string transformation operations are widely used to conceal different malicious intents. The 'privilege_escalation' and 'information_gathering' clusters further reflect privilege-related operations and environment reconnaissance.

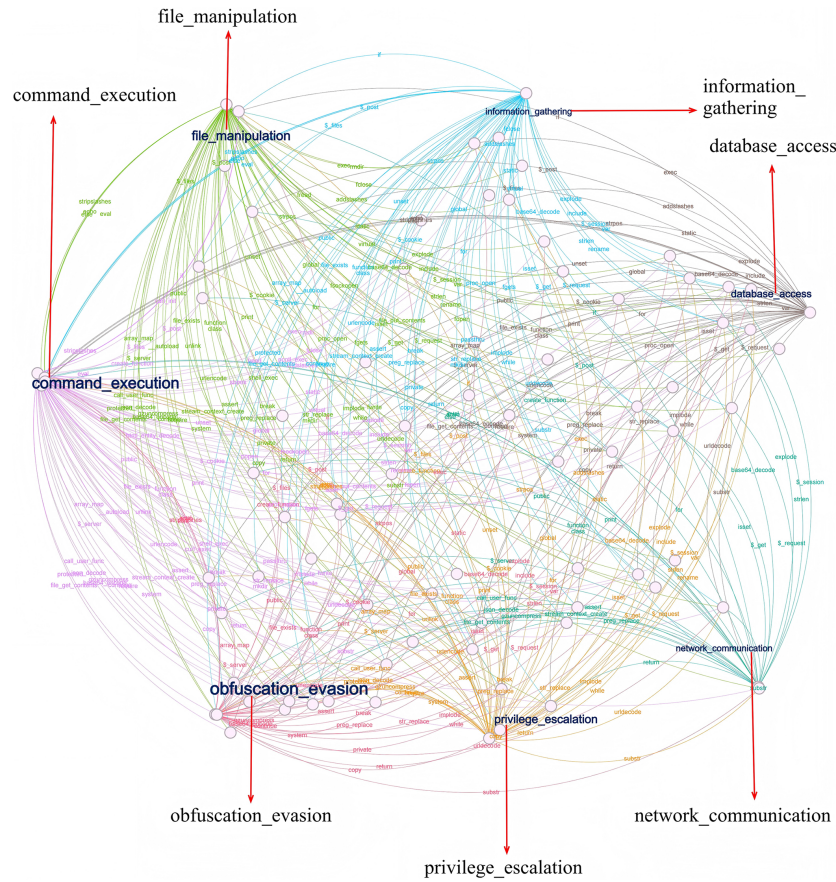


Figure 7: Extracting semantic token feature relationships of seven malicious behaviors via the relationship graph.

Overall, the relationship graph shows that MVFN-FG can capture semantic associations among fine-grained WebShell behaviors. By visualizing key tokens and cross-behavior interaction paths, the graph provides interpretable evidence for the effectiveness of multi-view feature extraction and fusion in WebShell malicious behavior analysis.

4.5 WebShell Large Model Deobfuscation Experiment Analysis (RQ4)

Since no established system summaries or directly related papers on comprehensive WebShell de-obfuscation comparison were retrieved, we compare several representative methods commonly used in the field to evaluate the effectiveness of our proposed approach. The LLM-based de-obfuscation method, driven by the GPT-4o model, is the core contribution of this work in terms of code restoration. For baseline comparisons, we selected String Restoration (SR), Variable and Function Restoration (VFR), Controller Restoration (CR), and Regular Expression Matching (REM).

To evaluate de-obfuscation performance, we use De-obfuscation Semantic Similarity (DSS) as a semantic similarity-based metric. The de-obfuscation module aims to recover the underlying semantics of obfuscated WebShell code, and DSS measures the semantic fidelity of the reconstructed results relative to their corresponding reference code. The metric is defined as the average semantic similarity between the reconstructed outputs and their corresponding reference code rather than as a binary de-obfuscation success rate. For each evaluated WebShell sample, the reference code is the corresponding original unobfuscated version retained before obfuscation. During construction of the de-obfuscation evaluation set, each

obfuscated sample is paired one-to-one with its original unobfuscated code, which preserves the intended execution logic and malicious functionality and therefore serves as the semantic reference for evaluating the reconstructed output.

To obtain the semantic representations used for similarity calculation, both the reference code and the reconstructed code are processed using the same WordShellVec representation procedure described in Section 3.4. Specifically, code semantic normalization is first applied to reduce non-semantic variations, after which the normalized code tokens are mapped into a continuous embedding space using the Skip-Gram framework. This process produces comparable semantic representations for the reference code and the corresponding de-obfuscated output. The De-obfuscation Semantic Similarity (DSS) is defined as follows:

$$DSS = \frac{1}{N} \sum_{i=1}^N \text{Sim}(\mathbf{y}_i, \hat{\mathbf{y}}_i) \times 100\%, \quad (7)$$

where N denotes the total number of evaluated WebShell samples, $\mathbf{y}_i$ denotes the semantic representation of the reference code for the i -th sample, and $\hat{\mathbf{y}}_i$ denotes the semantic representation of the corresponding de-obfuscated output. $\text{Sim}(\cdot, \cdot)$ denotes cosine similarity. For the DSS evaluation, the criterion is semantic fidelity relative to the corresponding reference code. A cosine similarity closer to 1 indicates stronger semantic preservation and better de-obfuscation performance, whereas a lower value indicates greater semantic deviation and poorer de-obfuscation performance. DSS is obtained by averaging the sample-level cosine similarities over the complete evaluation set and expressing the result as a percentage. No similarity threshold is used to convert the sample-level similarities into binary success or failure labels.

To further validate the effectiveness of the proposed method, we extend the comparison by incorporating several representative open-weight large language models, including Llama-2-7B, CodeLlama-7B, DeepSeek-Coder-6.7B, and Qwen2.5-Coder-7B. These models are evaluated under the same input sanitization and prompt formulation pipeline as GPT-4o to ensure a fair comparison. Specifically, all models receive the same prompt content with the unified structure $P = (R, T, C, O)$ and are instructed to return only de-obfuscated code, thereby reducing potential differences caused by prompt design. All open-weight models are evaluated individually without task-specific fine-tuning. They are loaded using 4-bit quantized inference with a batch size of 1. The maximum input length is limited to 4096 tokens for all evaluated models. No additional task-specific decoding optimization is applied. For GPT-4o API-based inference, we set temperature = 0.2, top_p = 0.9, and max_tokens = 2048. As illustrated in Fig. 8, the LLM-based method achieves the highest DSS of 78.90%, outperforming both traditional baselines and open-weight LLMs. Specifically, it surpasses the best-performing open-weight model (DeepSeek-Coder-6.7B, 76.80%) by 2.10 percentage points, while CodeLlama-7B, Qwen2.5-Coder-7B, and Llama-2-7B achieve 73.50%, 74.60%, and 70.20%, respectively. Traditional methods such as REM, VFR, and SR yield DSS values of 57.10%, 52.60%, and 45.30%, respectively. These results indicate that the proposed LLM-based method achieves the highest average semantic fidelity between the reconstructed outputs and their corresponding reference code.

To further ensure the reliability of the evaluation results, we conduct a manual verification procedure on a randomly sampled subset of 10% of the generated de-obfuscated samples. The manual inspection follows two criteria: (i) syntactic validity, ensuring executable correctness; (ii) runtime consistency, where both original and reconstructed code are executed in a sandbox environment to validate behavioral equivalence under identical conditions. To reduce the influence of sampling randomness, the above manual verification procedure is independently repeated four times using different random draws while maintaining the same 10% sampling ratio. For each sampled output, a reconstructed sample is regarded as successfully recovered only when both syntactic validity and runtime consistency are satisfied. The syntactic-validity pass rate and runtime-consistency pass rate are calculated separately for each random draw and then averaged across the

four rounds. This manual success criterion is used only for the supplementary verification procedure and does not participate in the calculation of DSS. Across the four independent random draws, the results remain stable, with an average syntactic validity pass rate of 91.3% and an average runtime consistency pass rate of 93.8%. The variation across the four rounds remains within 1.7%, indicating that the manual verification results are relatively insensitive to sampling randomness. These results provide complementary evidence that the reconstructed outputs generally preserve both executable syntax and runtime behavior, while remaining independent of the semantic-similarity-based DSS evaluation.

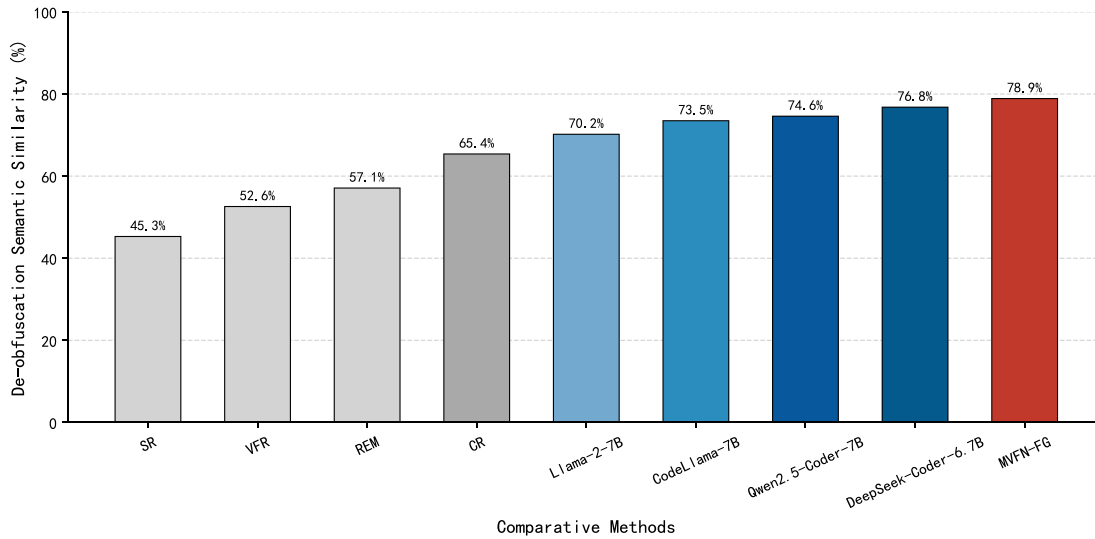


Figure 8: A comparison chart of various methods for deobfuscation.

In contrast, traditional methods focusing on lexical or syntactic restoration exhibit limited generalization ability. The integration of semantic-level reconstruction and structured constraints enables more stable recovery of execution logic, providing a reliable foundation for downstream intrusion detection tasks within the MVFN-FG framework.

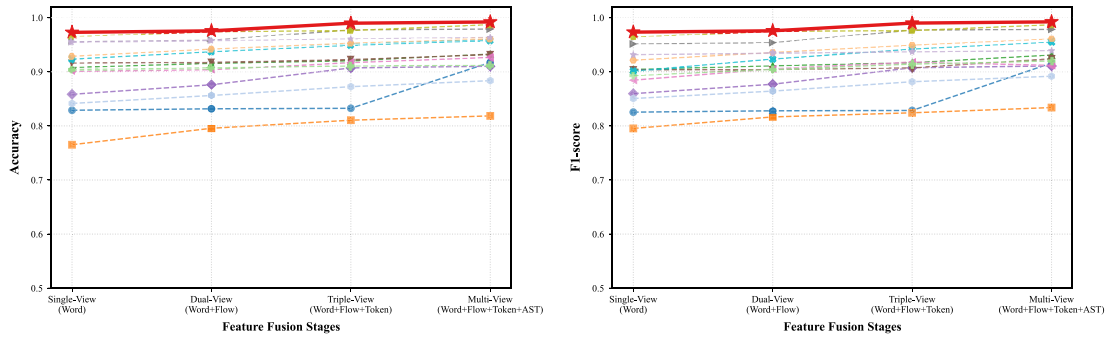
4.6 Ablation Study

To validate MVFN-FG, we conduct an ablation study on WebShell malicious behavior detection by progressively integrating WordShell, FlowShell, TokenShell, and ASTShell views. As shown in Table 6 and Fig. 9, most models achieve improved performance as additional feature views are incorporated, indicating that different feature views provide complementary information. Representative comparison methods also show that progressive feature fusion improves detection performance. Among conventional and deep learning baselines, LR improves its F1-score by 2.13%, 2.89%, and 3.86% from single-view to dual-view, triple-view, and full multi-view settings, respectively. CNN + BiLSTM, TextRank [27], and TF-IDF [7] also show consistent gains across the three fusion stages, confirming the effectiveness of progressive multi-view integration. In addition, ResNet-BiLSTM-Transformer (Obfuscated) shows consistent F1-score improvements after progressive feature fusion, while ResNet-BiLSTM-Transformer (Concat) achieves smaller but stable gains. MVFN-FG consistently achieves the best performance across all ablation settings. Compared with its single-, dual-, and triple-view settings, the full MVFN-FG improves Accuracy by 1.92%, 1.64%, and 0.23%, and improves F1-score by 1.90%, 1.65%, and 0.23%, respectively. These ablation results demonstrate that the progressive integration of WordShell, FlowShell, TokenShell, and ASTShell effectively enhances WebShell malicious behavior detection.

Table 6: Ablation study on WebShell malicious behavior detection.

Model	Single-View		Dual-View Fusion		Triple-View Fusion		Multi-View	
	WordShell		WordShell, FlowShell		WordShell, FlowShell, TokenShell		WordShell, FlowShell, TokenShell, ASTShell	
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score
KNN	0.8289	0.8254	0.8316	0.8279	0.8325	0.8285	0.9174	0.9162
LR	0.7652	0.7953	0.7956	0.8166	0.8105	0.8242	0.8185	0.8339
RF	0.9081	0.9047	0.9156	0.9111	0.9204	0.9169	0.9327	0.9309
SVM	0.8584	0.8596	0.8760	0.8770	0.9072	0.9075	0.9103	0.9108
CNN	0.9163	0.9037	0.9175	0.9042	0.9226	0.9074	0.9316	0.9241
BiLSTM	0.9011	0.8845	0.9039	0.9053	0.9169	0.9180	0.9265	0.9118
CNN + BiLSTM	0.9553	0.9511	0.9582	0.9535	0.9774	0.9771	0.9791	0.9784
CNN + BiLSTM + Transformer	0.9644	0.9637	0.9740	0.9747	0.9763	0.9763	0.9870	0.9868
TextRank ([27])	0.9233	0.9016	0.9368	0.9235	0.9485	0.9418	0.9570	0.9544
DNN ([29])	0.8415	0.8506	0.8562	0.8643	0.8725	0.8816	0.8836	0.8918
Tfidf ([7])	0.9286	0.9214	0.9418	0.9356	0.9527	0.9489	0.9600	0.9604
ResNet-BiLSTM-Transformer (Obfuscated)	0.9042	0.8931	0.9075	0.9046	0.9108	0.9142	0.9122	0.9200
ResNet-BiLSTM-Transformer (Concat)	0.9539	0.9316	0.9568	0.9342	0.9605	0.9367	0.9633	0.9389
MVFN-FG	0.9729	0.9733	0.9757	0.9758	0.9898	0.9900	0.9921	0.9923

We further evaluate the computational efficiency of MVFN-FG under the same experimental environment. The model contains 1,975,522 trainable parameters, with a saved model size of 7.57 MB. The total training time is 64.60 s, corresponding to 1.08 min, and the average epoch time is 10.75 s. The peak training and inference GPU memory consumption are 279.18 and 96.20 MB, respectively. The average batched inference time is 0.1386 ms per sample, with a throughput of 7216.77 samples/s. The average single-sample local detection latency is 6.1959 ms per sample, and the P95 latency is 8.2743 ms per sample. When the GPT-4o API-based de-obfuscation module is included, the end-to-end detection latency can be expressed as $T_{API} + 0.0061959$ s per sample, where T_{API} denotes the API latency caused by request transmission, cloud-side scheduling, response generation, and script length. The measured T_{API} was approximately 0.13–0.16 s per sample. After including the average local detection latency of 0.0061959 s, the complete pipeline latency was approximately 0.136–0.166 s per sample. Overall, MVFN-FG has a compact model size and low local inference overhead.

**Figure 9:** Impact of progressive feature view fusion on detection performance.

Note: ● denotes KNN, ■ denotes LR, ▲ denotes RF, ◆ denotes SVM, ▼ denotes CNN, ◆ denotes BiLSTM, ► denotes CNN + BiLSTM, ◆ denotes CNN + BiLSTM + Transformer, × denotes TextRank [27], ● denotes DNN [29], ● denotes Tfidf [7], + denotes ResNet-BiLSTM-Transformer (Obfuscated), * denotes ResNet-BiLSTM-Transformer (Concat), and ★ denotes MVFN-FG.

5 Conclusion

With the rise of sophisticated WebShell attacks, traditional detection methods struggle with multi-language heterogeneity, complex code obfuscation, and fine-grained behavior understanding. To address these challenges, we propose MVFN-FG, an advanced WebShell detection framework based on LLM-driven deobfuscation and multi-view feature fusion. MVFN-FG leverages the semantic reasoning capability of large language models to restore the execution logic of obfuscated scripts and introduces four complementary feature representations, namely WordShell, TokenShell, ASTShell, and FlowShell. These features are further processed by a hybrid ResNet-BiLSTM-Transformer model to capture hierarchical semantic relationships and global behavioral dependencies. Experimental results show that MVFN-FG not only distinguishes malicious WebShell samples from benign scripts, but also performs fine-grained malicious behavior classification across multiple WebShell attack categories. In the multi-view setting, MVFN-FG achieves an Accuracy of 0.9921 and an F1-score of 0.9923, outperforming existing baseline methods. Systematic experiments further demonstrate that MVFN-FG can effectively extract key behavioral characteristics and semantic correlations from WebShell payloads, providing practical technical support for WebShell protection and intelligent threat governance.

In future work, we plan to integrate the local detection component of MVFN-FG into target IoT devices or online firewalls to detect malicious behaviors of WebShells in real-world network environments. In addition, we will further construct a larger multilingual WebShell dataset covering more scripting languages, obfuscation strategies, and real-world attack scenarios to improve the generalization ability of the model. We also plan to investigate lightweight model compression, online updating, continual learning, and local or distilled LLM alternatives, aiming to improve the practicality of MVFN-FG for resource-constrained IoT environments.

Acknowledgement: Not applicable.

Funding Statement: This research was supported in part by the National Science Foundation of China (No.U24A20241, No.62202118, No.62562012, No.62361010, No.72261004); Major Scientific and Technological Special Project of Guizhou Province ([2024]003, [2024]014); Guizhou Provincial Basic Research Program (Natural Science) (Grant Qiankehe Foundation MS[2025]686, MS[2026]159, ZK[2024]652); Scientific and Technological Research Projects from Guizhou Education Department (Qian jiao ji [2023]003); Provincial Department of Science and Technology's Hundred level Innovation Talents Project (Guizhou Science and Technology Cooperation Platform Talents-GCC [2023] 018); Guizhou Provincial Key Technology R&D Program (PA[2025]004); Research Project for Recruited Talents at Guizhou University (GDRJH[2024]15); Scientific and Technological Innovation Talent Team of Guizhou Province for Cybersecurity Protection of Computing Power Network (CXTD[2025]029); Chongqing Key Laboratory of Public Big Data Security Technology (CQKL-QJ202300001); Science and Technology Program of GuiYang(ZK[2024]-1-2); Scientific and Technological Innovation Platform Research Project of Guizhou Province (CXPTXM[2025]024).

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Di Yao; methodology, Di Yao, Yuling Chen and Xiuzhang Yang; validation, Yuling Chen and Xiuzhang Yang; formal analysis, Xiuzhang Yang; investigation, Di Yao; resources, Di Yao; data curation, Di Yao; writing—original draft preparation, Di Yao; writing—review and editing, Di Yao; supervision, Xuewei Wang, Haiwei Sang, Zhi Ouyang and Weijie Tan; project administration, Haiwei Sang; funding acquisition, Zhi Ouyang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The raw datasets used in this study are openly available in the following GitHub repositories: <https://github.com/tanjiti/webshellSample>, <https://github.com/JohnTroony/php-webshells>, <https://github.com/LandGrey/webshell-detect-bypass>, <https://github.com/China-Eugene/webshell>, and <https://github.com/HziecC/WebshellDetectDataset>. The processed and derived data generated in this study, including the preprocessed samples

and stratified training/test splits, are not publicly available but may be obtained from the corresponding author upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A Unified Prompt Template for LLM-Based WebShell Deobfuscation

The template for WebShell deobfuscation prompt words is shown in Fig. A1.

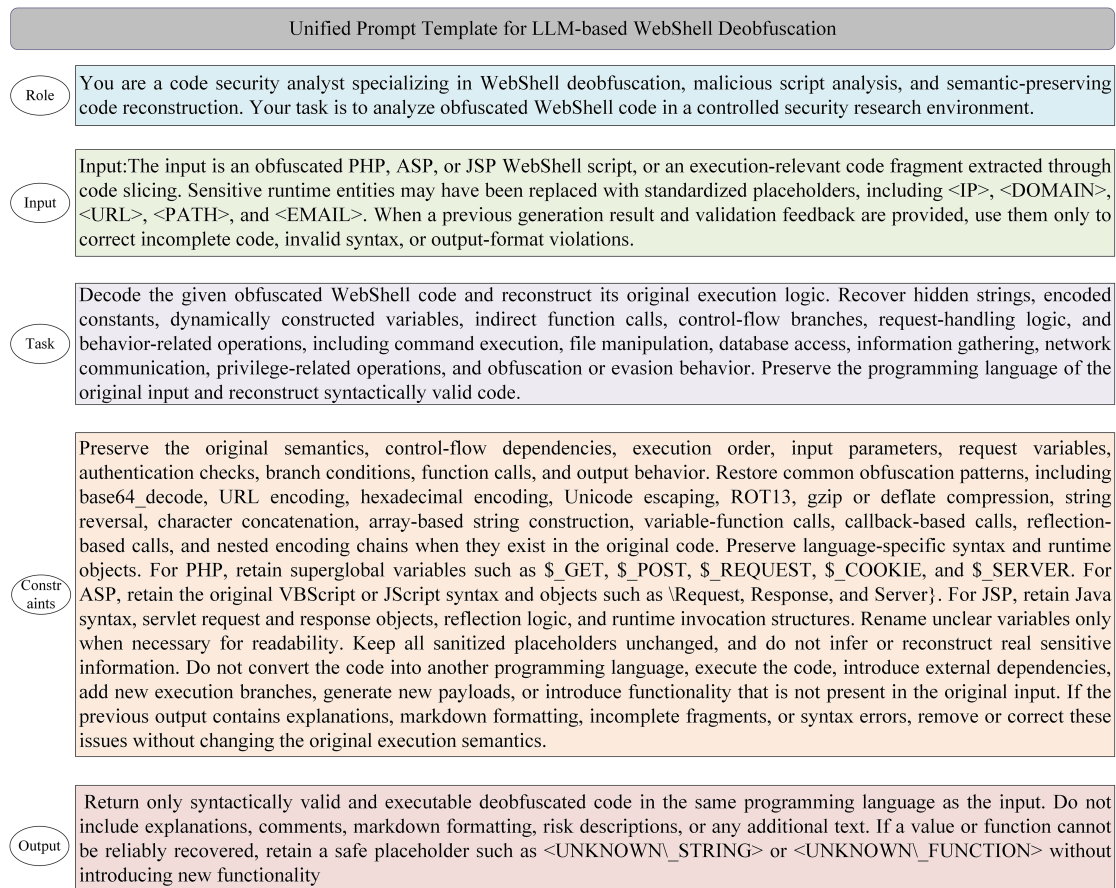


Figure A1: Unified prompt template for LLM-based WebShell deobfuscation.

References

1. Sikder AK, Petracca G, Aksu H, Jaeger T, Uluagac AS. A survey on sensor-based threats and attacks to smart devices and applications. *IEEE Commun Surv Tutor*. 2021;23(2):1125–59. doi:10.1109/comst.2021.3064507.
2. Zscaler ThreatLabz. Zscaler ThreatLabz 2023 enterprise IoT and OT threat report [Internet]. Zscaler; 2023 [cited 2026 Aug 27]. Available from: <https://info.zscaler.com/resources/industry-reports/threatlabz-2023-enterprise-iot-ot-threat-report>.
3. Zhou A, Luktarhan N, Ai Z. Research on WebShell detection method based on regularized neighborhood component analysis (RNCA). *Symmetry*. 2021;13(7):1202. doi:10.3390/sym13071202.
4. Alasmay H, Anwar A, Abusnaina A, Alabduljabbar A, Abuhamad M, Wang A, et al. ShellCore: automating malicious IoT software detection using shell commands representation. *IEEE Internet Things J*. 2021;9(4):2485–96.

5. Dong C, Li D. AST-DF: a new WebShell detection method based on abstract syntax tree and deep forest. *Electronics*. 2024;13(8):1482.
6. Qureshi SU, He J, Tunio S, Zhu N, Nazir A, Wajahat A, et al. Systematic review of deep learning solutions for malware detection and forensic analysis in IoT. *J King Saud Univ Comput Inf Sci*. 2024;36(8):102164. doi:10.1016/j.jksuci.2024.102164.
7. NM MB, Kannaiah SK. Effective WebShell detection in IoT networks employing TF-IDF, Word2Vec and SVM. In: *Proceedings of the 2024 IEEE International Conference of Electron Devices Society Kolkata Chapter (EDKCON)*; 2024 Nov 23–24; Kolkata, India. p. 1–6.
8. Zhu T, Weng Z, Fu L, Ruan L. A web shell detection method based on multiview feature fusion. *Appl Sci*. 2020;10(18):6274. doi:10.3390/app10186274.
9. Guo Y, Marco-Gisbert H, Keir P. Mitigating WebShell attacks through machine learning techniques. *Future Internet*. 2020;12(1):12. doi:10.3390/fi12010012.
10. 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 Commun*. 2023;20(11):202–24. doi:10.23919/jcc.fa.2022-0509.202311.
11. Li R, Zhang C, Chai H, Ying L, Duan H, Tao J. PowerPeeler: a precise and general dynamic deobfuscation method for PowerShell scripts. In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*; 2024 Oct 14–18; Salt Lake City, UT, USA. p. 4539–53.
12. Zhang S, Li S, Lu J, Yang W. PowerASTNN: a deobfuscation and AST neural network enabled effective detection method for malicious PowerShell scripts. *Comput Secur*. 2025;154:104441.
13. Yonamine S, Taenaka Y, Kadobayashi Y. Tamer: a sandbox for facilitating and automating IoT malware analysis with techniques to elicit malicious behavior. In: *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP)*; 2022 Feb 9–11; Virtual. p. 677–87.
14. Nazir A, He J, Zhu N, Wajahat A, Ma X, Ullah F, et al. Advancing IoT security: a systematic review of machine learning approaches for the detection of IoT botnets. *J King Saud Univ Comput Inf Sci*. 2023;35(10):101820.
15. Nazir A, He J, Zhu N, Wajahat A, Ullah F, Qureshi S, et al. Collaborative threat intelligence: enhancing IoT security through blockchain and machine learning integration. *J King Saud Univ Comput Inf Sci*. 2024;36(2):101939.
16. Gupta GP, Kumar P, Aljuhani A. An efficient malware detection framework for enhancing software security in resource-constrained systems. *IEEE Internet Things J*. 2025;12(21):44897–908. doi:10.1109/jiot.2025.3596313.
17. He C, Fan H, Yin L, Yan H, Luo X, Li H, et al. MalAE: a feature-optimized and autoencoder ensemble-based method for IoT malware classification. *IEEE Internet Things J*. 2025;12(14):27181–92.
18. Wei L, Wang Y, Li X, Li J, Huang Y, Liu Z. A detection method for malware communication traffic via encrypted traffic analysis. *IEEE Internet Things J*. 2025;12(16):33482–92. doi:10.1109/jiot.2025.3577245.
19. Wu W, Peng H, Zhu H, Li L. MVC-RSN: a malware classification method with variant identification ability. *IEEE Internet Things J*. 2024;11(21):35654–68.
20. Zhang H, Hu X, Shen Y, Xu H, Shen S, Li R. Mitigating malware propagation in social internet of things using an exact Markov chain-based epidemic method. *IEEE Internet Things J*. 2025;12(13):24104–18. doi:10.1109/jiot.2025.3554230.
21. Shen Y, Shepherd C, Ahmed CM, Yu S, Li T. Comparative DQN-improved algorithms for stochastic games-based automated edge intelligence-enabled IoT malware spread-suppression strategies. *IEEE Internet Things J*. 2024;11(12):22550–61. doi:10.1109/jiot.2024.3381281.
22. Pan Z, Chen Y, Chen Y, Shen Y, Guo X. WebShell detection based on executable data characteristics of PHP code. *Wirel Commun Mob Comput*. 2021;2021(1):5533963. doi:10.1155/2021/5533963.
23. Yang X, Chen Y, Chen C, Yao D, Li C, Zhou Y, et al. XLM4Detector: multi-stage deobfuscation and semantic-driven excel 4.0 macro malware detection. *IEEE Internet Things J*. 2025;12(16):33271–92.
24. Anggraini D, Girsang AS. WebShell detection based on bytecode feature with convolutional neural network. *J Theor Appl Inf Technol*. 2023;101(18):7443–54.

25. Tu TD, Guang C, Xiaojun G, Wubin P. WebShell detection techniques in web applications. In: Proceedings of the Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT); 2014 Jul 11–14; Hefei, China. p. 1–7.
26. Ma M, Han L, Zhou C. Research and application of artificial intelligence based WebShell detection model: a literature review. arXiv:2405.00066. 2024.
27. Lee HJ, Hwang SJ, Pratiwi M, Choi YH. Obfuscated PHP WebShell detection using the WebShell tailored TextRank algorithm. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing; 2024 Apr 8–12; Avila, Spain. p. 1358–65.
28. Tsai MH, Lin CC, He ZG, Yang WC, Lei CL. PowerDP: de-obfuscating and profiling malicious PowerShell commands with multi-label classifiers. IEEE Access. 2022;11:256–70.
29. Hannousse A, Nait-Hamoud MC, Yahiouche S. A deep learner model for multi-language WebShell detection. Int J Inf Secur. 2023;22(1):47–61. doi:10.1007/s10207-022-00615-5.
30. Fang Y, Qiu Y, Liu L, Huang C. Detecting WebShell based on random forest with FastText. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence; 2018 Mar 12–14; Chengdu, China. p. 52–6.
31. Wu Y, Sun Y, Huang C, Jia P, Liu L. Session-based WebShell detection using machine learning in web logs. Secur Commun Netw. 2019;2019(1):3093809. doi:10.1155/2019/3093809.
32. Yong B, Wei W, Li KC, Shen J, Zhou Q, Wozniak M, et al. Ensemble machine learning approaches for WebShell detection in internet of things environments. Trans Emerg Telecommun Technol. 2022;33(6):e4085. doi:10.1002/ett.4085.
33. Yang W, Sun B, Cui B. A WebShell detection technology based on HTTP traffic analysis. In: Proceedings of the International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing; 2018 Jul 4–6; Matsue, Japan. p. 336–42.
34. Tian Y, Wang J, Zhou Z, Zhou S. CNN-WebShell: malicious web shell detection with convolutional neural network. In: Proceedings of the 2017 VI International Conference on Network, Communication and Computing; 2017 Dec 8–10; Kunming, China. p. 75–9.
35. Lv ZH, Yan HB, Mei R. Automatic and accurate detection of WebShell based on convolutional neural network. In: Proceedings of the China Cyber Security Annual Conference; 2018 Aug 14–16; Beijing, China. p. 73–85.
36. Le HV, Nguyen TN, Nguyen HN, Le L. An efficient hybrid WebShell detection method for webserver of marine transportation systems. IEEE Trans Intell Transp Syst. 2021;24(2):2630–42. doi:10.1109/tits.2021.3122979.
37. Rose AJ, Kabban CMS, Graham SR, Henry WC, Rondeau CM. Malware classification through abstract syntax trees and L-moments. Comput Secur. 2025;148(null):104082. doi:10.1016/j.cose.2024.104082.
38. Wang Z, Yang J, Dai M, Xu R, Liang X. A method of detecting WebShell based on multi-layer perception. Acad J Comput Inf Sci. 2019;2(1):81–91.
39. Shang M, Han X, Zhao C, Cui Z, Du D, Jiang B. Multi-language WebShell detection based on abstract syntax tree and TreeLSTM. In: Proceedings of the 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD); 2024 May 8–10; Tianjin, China. p. 377–82.
40. Xie B, Li Q, Wang Y. PHP-based malicious WebShell detection based on abstract syntax tree simplification and explicit duration recurrent networks. Comput Secur. 2024;146(1):104049. doi:10.1016/j.cose.2024.104049.
41. Feng P, Wei D, Li Q, Wang Q, Hu Y, Xi N, et al. GlareShell: graph learning-based PHP WebShell detection for web server of industrial internet. Comput Netw. 2024;245:110406.
42. Li T, Ren C, Fu Y, Xu J, Guo J, Chen X. WebShell detection based on the word attention mechanism. IEEE Access. 2019;7:185140–7. doi:10.1109/access.2019.2959950.