SCORE: Syntactic Code Representations for Static Script Malware Detection

1st Ecenaz Erdemir
Amazon Web Services
New York, NY, USA
ecenaz@amazon.com

2nd Kyuhong Park
Amazon Web Services
New York, NY, USA
kyuhongp@amazon.com

3rd Yi Fan
Amazon Web Services
New York, NY, USA
fnyi@amazon.com

***Abstract*—Cloud technology adoption has intensified the impact of server-side script attacks, exposing organizations to greater risks against system and data integrity. These server-side scripts in languages like Python, Perl and Bash that operate on server runtime environments can steal data, compromise credentials, and disrupt operations. Unlike executables with standardized formats (e.g., ELF, PE), scripts are plain-text files with diverse syntax, making them harder to detect using traditional methods. To address this, we propose a novel static script malware detection approach that combines feature extraction from abstract syntax trees (ASTs) with deep learning (DL) models, targeting server-side threats. ASTs capture the hierarchical syntactic structure of code, enabling richer feature representations. We then propose both sequential and graph-based models that exploits these feature representations to detect script malware. Our evaluation on over 400K Bash, Python, and Perl scripts—using 90K for training, validation and testing, and the rest for imbalanced tests—shows that our method achieves up to 81% higher true positive rate than leading signature-based antivirus solutions while maintaining a low false positive rate (0.17%). Our approach also outperforms various neural network–based detectors, demonstrating strong capability in learning and identifying malicious code behavior.**

***Index Terms*—Script malware, malicious code, abstract syntax tree, deep learning, recurrent and graph neural networks.**

1. Introduction

Script-based malware has become a dominant attack vector against Linux-based cloud environments, leveraging the versatility and portability of server-side scripting languages such as Python, Perl, and Bash. These malicious scripts act either as standalone threats (e.g., ransomware, cryptominers, backdoors) or as components in multi-stage attacks that enable payload delivery and execution. Comparable to traditional executables in functionality, script malware is often harder to detect using static signatures or classic ML models due to code obfuscation and language diversity.

Recent Python-based credential harvesters, such as Legion and AlienFox, exemplify the threat: both target AWS console credentials and abuse cloud resources for spamming, phishing, or privilege escalation [1], [2]. The prevalence of script malware has surged—doubling since 2017 and accounting for nearly 40% of cyberattacks by 2020 [3], with another doubling by early 2024 [4]. Given their increasing role in cloud abuse [5]–[7], effective and scalable script malware detection is a pressing security challenge.

Static analysis, which inspects files without execution unlike dynamic analysis, remains an attractive approach for scalability, safety, and speed [8]. Yet, signature-based methods are easily evaded through renaming, control-flow reshaping, or value obfuscation [8], [9]. Representation learning–based approaches can generalize beyond handcrafted signatures [10]–[15], but existing models either rely on raw byte-level inputs [16], [17] or incur excessive cost when using large language models (LLM). This creates a gap for efficient, robust detectors that capture invariant structural semantics of malicious code rather than surface-level features.

Script malware analysis also introduces unique challenges: the need for language-specific parsing, limited structured information in byte-level features compared to executables, and high complexity in advanced code representations. To address these, we propose feature extraction and detection methods tailored to server-side scripts.

We investigate three research questions:

- **RQ1**: How can malicious patterns in server-side scripts be effectively represented as features for scalable detections?
- **RQ2**: Can scalable deep learning (DL) models capture the complex structural characteristics of scripts, and which architectures are most effective?
- **RQ3**: How does our best-performing DL approach compare with conventional rule-based detectors and state-of-the-art DL or LLM-based models in detecting real-world threats and providing threat coverage?

To this end, we introduce SCORE-T, an abstract syntax tree (AST)-based feature extraction method that parses scripts into hierarchical syntactic tree structures and combines them with raw byte-strings for richer representations. We design two detection models: (1) a Sequential Model (SM) that is composed of convolutional neural network (CNN) and bidirectional long short-term memory (bi-LSTM) layers to learn hierarchical embeddings from serialized SCORE-T features, and (2) a Graph Representation Learning (GRL) model that derives graph embeddings via similarity learning with a machine learning (ML)-based detector (e.g., XGBoost [18]). Figure 1 shows an overview.

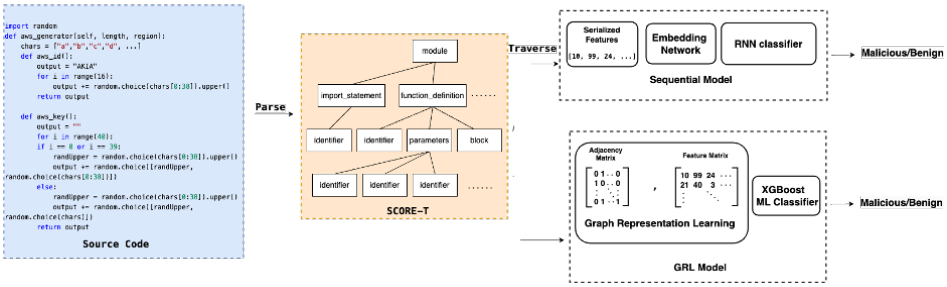

Figure 1: System overview: Our system first parses scripts for their SCORE-T features. ASTs are either traversed and fed to the SM or fed directly to a GRL model along with their adjacency matrix to output a malicious or benign verdict.

We benchmark against commercial and open-source anti-virus (AV) programs, byte-level detectors [17], AST-based sequential and graph convolutional network (GCN) models [11], and a CodeBERT-based detector [13]. Our evaluations show that (i) serialization order of ASTs matters—SM with breadth-first traversal (BFT) consistently outperforms baselines, while GRL enhances detection under threat-labeled and depth-first traversal (DFT) conditions; and (ii) overall, our approaches surpass AVs, byte-level, AST-based, and token-based detectors.

Our contributions can be summarized as follows:

1) We introduce SCORE-T, a novel AST-based feature extraction framework for server-side scripts, integrated into both sequential and graph-based models. Unlike prior AST detectors that rely solely on syntactic node embeddings, SCORE-T fuses byte-token features together with AST nodes, learning obfuscation-resilient representations. We further propose a threat-aware contrastive objective that clusters samples by threat family for the graph approach, improving generalization to unseen variants.
2) We curate a large, diverse dataset of benign and malicious scripts and conduct extensive evaluations showing that our models:
   - Substantially outperform commercial and open-source AVs, byte-level ML detectors, AST-based baselines, and a CodeBERT-based detector.
   - Achieve over 95% coverage of high-priority threats, including cryptominers, ransomware, credential stealers.

2. Motivation and Scope

We motivate our work through real-world script malware. The Python-based credential harvester Legion exploits web servers running CMS or frameworks such as Laravel, using regex-based extraction to steal credentials from email providers, cloud platforms (e.g., AWS), and payment systems [1]. It specifically targets AWS console, SNS, S3, and SES credentials, then abuses compromised infrastructure for mass spamming and phishing. AlienFox performs similar operations—harvesting API keys and secrets, maintaining AWS persistence, collecting quotas, and automating spam via victim accounts [2]. Detecting such scripts is critical to cloud security. Signature-based scanners, which rely on surface patterns, are easily bypassed through function renaming, library substitution, or obfuscation. Figure 1 shows a Legion function that brute-forces AWS IDs and keys. While pattern matching fails under such modifications, AST-based features remain stable because they capture the underlying code structure in the form of a tree with functional node naming. We leverage these stable structural features through DL to identify malicious script behaviors statically and efficiently.

Our scope is ML-based static analysis of server-side scripts in cloud environments—specifically Python, Perl, and Bash. By "server-side scripts," we mean programs executed on servers to process requests, manage data, or perform automation tasks. Unlike prior work focusing on client-side languages (e.g., JavaScript, PHP, HTML), we target the portable, Linux-prevalent server-side languages that underpin cloud infrastructure. Our approach focuses on code-level structure, hence does not address supply chain attacks (e.g., 2021 PyPI compromise [19]), where malicious packages are distributed through legitimate repositories. Such attacks are not intrinsic to the code and are out of scope [20], [21].

3. Syntactic Code Representations (SCORE)

Static analysis of script malware benefits from mature parsing tools originally developed for compilers. These tools construct ASTs from formal grammars, efficiently capturing a program's hierarchical structure. We repurpose such parsing library as tokenizer and feature extractor for ML-based malware detection, enabling structured code understanding beyond surface-level tokens. Our AST-based feature extractor, SCORE-T, implements this concept (Figure 2). We focus on server-side scripting languages—Python, Perl, and Bash/Shell—which dominate cloud malware and execute directly on target systems. Compiled languages such as C or Java are out of scope, as they typically appear as binaries rather than interpretable source code on victim machines.

3.1. Abstract Syntax Tree Features (SCORE-T)

ASTs encode code syntax as hierarchical node–edge structures, forming the backbone of compilers, linters, and analyzers. For malware detection, we adapt the Tree-sitter parser [22] for its speed, robustness, and language coverage.

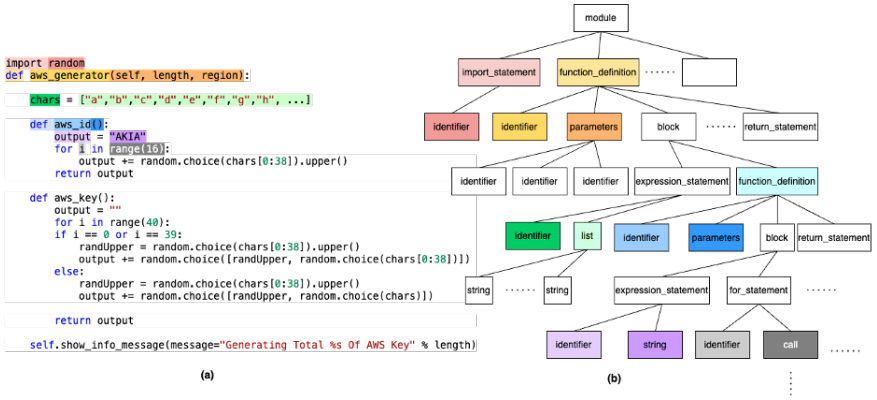

Figure 2: (a) Python script from Legion, a credential harvester, and (b) its AST example.

Tree-sitter uses context-free regex grammars to parse code into ASTs, tokenizing text into named nodes (Figure 2a). For example, in `import base64`, the `import` keyword forms an `import statement` node, while `base64` appears as an `identifier` leaf branching from that node. Thus ASTs nest functionally-related chunks of code, such as function definitions and control-flow statements, into subtrees. We further standardize node names into a cross-language vocabulary, following [11]. For instance, the root nodes `program` (Bash), `module` (Python), and `source file` (Perl) all denote the code file. Code blocks show similar variation, which we unify to condense the AST features and permit cross-language inferences.

We parse each script into an AST and couple each node with its associated raw code (byte-strings), and use this graph data structure as features in two ways: (*i*) traversing the AST with depth-first or breadth-first search to generate sequences of node names and code bytes for sequential models (Section 4.1), or (*ii*) preserving the tree for direct use in graph-based models (Section 4.2).

4. Malware Detection

In this section, we detail two models that classify code as malicious or benign using our AST-based representations: the SM –designed for serialized SCORE-T features after tree traversal–, and GRL –operating directly on the hierarchical tree structure without serialization using graph similarity learning. Each model leverages the distinct structural characteristics of code representations to detect malicious behavior.

4.1. Sequential model (SM)

The sequential script malware detector is a CNN and bi-LSTM model for the joint byte-syntactic features of serialized SCORE-T representations. It is composed of two modules: one for generating embeddings of the joint byte-syntactic features, one for generating malicious and benign verdicts from the resulting sequences of feature embeddings. The embedding module fits two embeddings: syntactic features (ASTs) and byte features, and concatenates them into a joint embedding per item in the sequence.

We designed a specialized CNN-based model adapted from [23] to generate embeddings of SCORE-T features, which can be serialized into sequences by breadth first traversal (BFT) or depth first traversal (DFT). To capture multiple level of meaning in the trained embeddings, we use three 1-D convolutional modules with filter dimensions $64, 128, 192$ and kernel sizes from 1 to 3 (covering different sizes of context) and three modules with kernel sizes from 6 up to 18 (covering three scopes/characters). Then, the concatenation of all convolutional module outputs is sent through a highway layer [24], which implements a self gating mechanism to control information flow and adds skip connections to prevent vanishing gradients. A plain feedforward neural network layer typically applies an affine transform followed by a nonlinear activation, $H(\cdot, \mathbf{W_H})$, to its input $\mathbf{x}$, e.g., $\mathbf{y} = H(\mathbf{x}, \mathbf{W_H})$, where $\mathbf{y}$ is the layer output and the transform H is parametrized by $\mathbf{W_H}$. On the other hand, a highway layer contains two additional nonlinear transforms $T(\mathbf{x}, \mathbf{W_T})$ and $C(\mathbf{x}, \mathbf{W_C})$ such that:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}). \quad (1)$$

We designed a similar CNN-based model to generate embeddings of the byte features. The only difference is we learn an embedding for each byte, followed by three 1-D convolutional modules with filter dimensions $64, 128, 192$ and kernel sizes $16, 32, 128$, respectively. These two embeddings are concatenated into a joint byte-syntactic embedding.

We model the resulting sequence of joint byte-syntactic SCORE embeddings using a bi-LSTM with 2 layers and hidden dimension of 256, followed by an attention module and an output dense layer to classify the script as malicious or benign. The model is trained by optimizing the binary cross entropy (BCE) loss between the true labels and predictions:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} \big(y_i \cdot \log(\hat{y}_i) + (1-y_i) \cdot \log(1-\hat{y}_i)\big), \quad (2)$$

where y_i is the true label of i^{th} sample and $\hat{y}_i$ is the prediction of SM for i^{th} sample. In our model, the bi-LSTM acts as a sequential classifier that detects global features that are informative of maliciousness. Overall, our approach utilizes

the unique structural characteristics of code representations to create a robust script malware detector. The combination of CNN and RNN allows for encoding of both local and global features, enabling our model to capture the complex relationships between bytes and code chunks.

4.2. Graph Representation Learning (GRL) Model

We propose a GRL-based detection model that embeds the tree structure of scripts into byte-syntactic representations. Adapting the graph similarity learning framework from [25], our model iteratively aggregates local structural information to learn embeddings where similar scripts (benign–benign or malicious–malicious) are close in vector space, while dissimilar pairs are separated, enabling effective classification of malicious behavior.

Each script's AST is treated as a graph $G = (V, E)$, where node $i \in V$ has a feature vector x_i, and edge $(i, j) \in E$ has a feature vector $x_{i,j}$. Since our SCORE-T features contain node-level byte-syntactic features but no edge attributes, we set $x_{i,j}$ to a constant vector of ones. Node features x_i are constructed by concatenating the syntax and byte features.

The model comprises an encoder, propagation layers, and an aggregator [25]. The encoder maps node and edge features into vector representations using multilayer perceptrons (MLPs): $\mathbf{h}_i^{(0)} = MLP_{node}(x_i)$ and $\mathbf{e}_{i,j} = MLP_{edge}(x_{i,j})$. During propagation, node embeddings are iteratively updated via message passing:

$$\mathbf{m}_{j\to i} = f_{message}(\mathbf{h}_i^{(t)}, \mathbf{h}_j^{(t)}, \mathbf{e}_{i,j}), \tag{3}$$

$$\mathbf{h}_i^{(t+1)} = f_{node}\Big(\mathbf{h}_i^{(t)}, \sum_{j:(j,i)\in E} \mathbf{m}_{j\to i}\Big), \tag{4}$$

where $f_{message}$ is an MLP and f_{node} a gated recurrent unit (GRU). The message aggregation can be sum, mean, max, or attention-based. After T iterations, each node $\mathbf{h}_i^{(T)}$ encodes its T-hop neighborhood. The final graph embedding $\mathbf{h}_G$ is computed via a gated aggregation:

$$\mathbf{h}_G = MLP_G\left(\sum_{i\in V} \sigma\big(MLP_{gate}(\mathbf{h}_i^{(T)})\big) \odot MLP(\mathbf{h}_i^{(T)})\right), \tag{5}$$

where gating weights filter irrelevant nodes, improving empirical performance. We compute the similarity between two graphs (G_1, G_2) using *Hamming similarity*, which outperformed cosine and Euclidean distance on our dataset. Hamming similarity is better suited for our use-case as it directly captures fine-grained structur differences, providing a more precise measure of structural change.

For contrastive learning, we define pairs based on label-wise (benign–malicious) and threat-wise (within threat families) relationships. Embeddings are constrained to binary vectors $\mathbf{h}_G \in \{-1, 1\}^H$ to support efficient nearest-neighbor search. We minimize the following pairwise loss:

$$L_{pair} = \mathbb{E}_{(G_1,G_2,\ell)}[(\ell - s(G_1, G_2))^2]/4, \tag{6}$$

where $s(G_1, G_2) = \frac{1}{H}\sum_{i=1}^{H} tanh(h_{G_1 i}) \cdot tanh(h_{G_2 i})$ is the approximate average Hamming similarity. The loss, bounded in $[0, 1]$, encourages positive pairs to have similarity near 1 and negative pairs near -1, offering greater stability than Euclidean-based contrastive losses. Once the embeddings are trained, we use an XGBoost [18] classifier for final malware detection based on the learned graph representations.

TABLE 1: Dataset summary.

Lang.	Total Dataset (412,257 scripts)						Balanced Subset (90,000 scripts)		
	Benign Set			Malicious Set					
	EC2	GitHub	Tot.	VT	HP.	Tot.	Train	Val.	Test
Python	56432	278658	335090	15991	99	16090	26816	2682	2682
Bash	7604	20054	27658	21767	9152	30919	46100	4608	4608
Perl	1005	245	1250	1125	125	1250	2084	208	208

5. Experimental Study

We evaluate our proposed script malware detection methods against three objectives: high accuracy, comparison with existing tools, and threat coverage.

1) **High Accuracy.** We measure True Positive Rate (TPR), False Positive Rate (FPR), Precision, F1 score, and AUROC. Our malicious scripts come from VirusTotal (VT) and our decade-long running industry scale honeypot (HP), while benign scripts are collected from newly provisioned cloud instances and GitHub repositories. This ensures diverse and realistic cloud environments (Section 5.1). Our methods target low FPR and high TPR regime, aligning with the low tolerance of the malware domain to FPs and False Negatives (FNs). We achieve up to 0.98 detection with an FPR of 0.00172 (Section 5.3.1). We also perform FP/FN analysis to identify common error patterns and assess their impact (Section 5.3.2).
2) **Comparison with Existing Tools.** We compare against signature-based AVs (BitDefender[1], ClamAV[2]) and DL-based approaches (MalConv2, SAST, GAST, UAST, CodeBERT+XGBoost). We demonstrate that AST-based features and our architectures outperform byte-level, AST-only, and foundation model embeddings (Sections 5.3.3, 5.3.4). Performance comparisons also include latency and throughput trade-offs.
3) **High Threat Coverage.** We evaluate coverage of common cloud threats using VT, BitDefender, and HP verification. Our top model, SM with SCORE-T (BFT), achieves near-full coverage for most threat families, including scripts missed by VT, while maintaining robustness against obfuscations like Base64 and XOR(Section 5.4).

This evaluation demonstrates that our approaches provide accurate, robust, and practical script malware detection in realistic cloud environments.

5.1. Dataset

We curated 400K Python, Bash, and Perl scripts into a large dataset, then randomly sampled 90K scripts (Table 1) splitting as 10:1:1 for train/validation/test, maintaining equal

1. https://www.bitdefender.com
2. https://www.clamav.net

numbers for benign and malicious scripts. We also performed imbalanced tests with the rest of benign set (310K) to mimic a realistic environment.

Benign scripts originate from newly provisioned cloud instances (EC2, Amazon Linux 2, Ubuntu) and GitHub repositories with $\geq$1000 stars.

Malicious scripts come from our in-house HP and from VT, detected between 2019–2024 and flagged by $\geq$10 AV vendors for high-confidence labels. The HP scripts that are not found on VT were clustered by fuzzy hashing and every cluster is manually verified and labeled by security engineers. The test set contains 3,750 scripts: 2,911 VT-reported (77.6%) and 839 manually verified (22.4%).

Properties: 40% of training files are $\geq$1.2MB (60% malicious). Malicious scripts often show high entropy ($\geq$9.5), reflecting obfuscation, encryption, or packing. Memory constraints cap AST nodes and byte-string length, while parsing treats obfuscated chunks as single entities, ensuring evaluation across diverse obfuscation scenarios.

5.2. DL Model Parameters

AST nodes are capped at 700 and byte-strings at 512 tokens to reduce memory usage. SM uses binary cross-entropy loss with Adam optimizer ($lr = 0.001$, weight decay 0.0001). GRL uses Hamming loss with Adam ($lr = 0.0001$, weight decay 0.00001) and 5 graph propagation steps. Experiments compare our models with BitDefender, ClamAV, and DL baselines: MalConv2 (byte-level and SCORE-T BFT), SAST, GAST, UAST, and XGBoost with CodeBERT (microsoft/codebert-base) embeddings of raw code. Hardware: Intel Xeon CPU + 4 Tesla V100 GPUs.

5.3. Evaluation and Results

Here, we present experimental results for our script malware detectors: SM with SCORE-T, GRL with SCORE-T, and their comparisons with 1) AVs, i.e., BitDefender and ClamAV, 2) DL-based detectors, i.e., MalConv2 [17], SAST, GAST and UAST [11], and 3) an LLM-based (CodeBERT) detector. Table 2 shows their performances on our dataset. Throughout evaluations, we refer to our approaches by their malware detectors and feature extractors: SM with SCORE-T (DFT), SM with SCORE-T (BFT), GRL with SCORE-T (label-wise) and GRL with SCORE-T (threat-wise).

5.3.1. Overall Detection Results. We evaluated our script malware detectors—SM with SCORE-T and GRL with SCORE-T—on the curated dataset described in Section 5.1. Table 2 summarizes performance metrics including F1 score, TPR, FPR, Precision, and AUROC for the balanced set. Across all proposed approaches, we observe high detection rates, with TPR consistently above 0.95 and FPR around 0.0017, demonstrating both effectiveness and reliability.

For SM, the AST traversal order significantly impacts performance. BFT (Breadth-First Traversal) achieves higher TPR than DFT (Depth-First Traversal), especially under the AST node cap of 700. BFT prioritizes capturing higher-level nodes, which often encode the overall script structure and early indicators of malicious behavior. This is particularly advantageous for large scripts where malicious actions occur later in the code, which DFT might truncate due to the cap.

TABLE 2: Performance results for ClamAV, BitDefender, MalConv2 [17], SAST, GAST and UAST [11] and the proposed models in terms of F1 score, FPR, TPR (Recall), Precision and AUROC score.

Detector	Feature	Variation	F1	FPR	Precision	TPR	AUROC
ClamAV	Files	–	0.28993	0	1	0.16954	–
BitDefender	Files	–	0.85590	0	1	0.74810	–
MalConv2	Byte-string	–	0.97016	0.00188	0.99813	0.94371	0.99756
MalConv2	SCORE-T	BFT	0.97350	0.00180	0.99810	0.95010	0.99784
SAST	AST	BFT	0.95276	0.00184	0.99791	0.91152	0.99781
GAST	AST	-	0.69786	0.00192	0.99642	0.53698	0.99443
UAST	AST	BFT	0.95967	0.00185	0.99813	0.92391	0.99690
CodeBERT XGBoost	CodeBERT embed.	–	0.96665	0.00192	0.99795	0.93725	0.99060
[Ours] **SM**	**SCORE-T**	DFT	0.97932	0.00180	0.99806	0.96128	0.99901
		BFT	**0.98949**	**0.00172**	**0.99825**	**0.98088**	**0.99919**
[Ours] GRL	SCORE-T	Label-wise	0.98175	0.00185	0.99807	0.96595	0.99901
		Threat-wise	0.98823	0.00185	0.99810	0.97855	0.99917

GRL leverages threat-wise contrastive learning to enhance graph-based embeddings. Unlike label-wise clustering, which only distinguishes between benign and malicious scripts, threat-wise clustering incorporates the semantic similarity of scripts belonging to the same threat family. This allows GRL to capture nuanced syntactic and structural patterns across scripts, resulting in improved TPR (0.97855) and overall AUROC compared to label-wise embeddings (0.96595 TPR). These results highlight the benefit of incorporating threat-specific structural similarity in representations.

More detail on the accuracy of our best model—SM with SCORE-T (BFT)— for balanced and imbalanced sets as well as PL coverage can be found in Appendix C.

5.3.2. False Positive and False Negative Analysis. We conducted an in-depth analysis of misclassifications to identify sources of error and opportunities for mitigation.

False Positives (FPs): Most FPs resulted from: 1) manually written scripts with long number sequences mimicking obfuscation patterns, and 2) non-English characters resembling encrypted/compressed code segments. These cases primarily affect detectors relying on byte-level features, while AST-based features capture syntactic structure over string content. This can be mitigated by adjusting feature weighting to balance AST structure and byte-string content, reducing misclassification in non-obfuscated benign scripts.

False Negatives (FNs): FNs mostly result from feature length limitations due to memory constraints, particularly when malicious behavior appears late in large scripts. This can be mitigated by using BFT traversal instead of DFT to capture overall script context first, increasing feature caps for more node/token processing, and implementing sliding window approaches for sequential embedding of large scripts. These collectively reduce missed malicious scripts across Python, Bash, and Perl, improving detection..

5.3.3. Comparison with AV Solutions. We benchmarked our models against signature-based antivirus (AV) engines,

ClamAV and BitDefender. Signature-based detection struggles with obfuscated or irregularly structured scripts, as it depends on fixed byte patterns rather than learned behavior. As shown in Table 2, our approach substantially outperforms AVs: BitDefender achieved F1 = 0.8559 (TPR = 0.7481) and ClamAV TPR = 0.1695, whereas SM with SCORE-T (BFT) reached F1 = 0.9895 and TPR = 0.9809 at FPR = 0.00172. The gap arises because commercial AVs are primarily tuned for Windows binaries and static indicators, whereas our model captures language semantics and behavioral context through AST-based features. This allows accurate detection of unseen and obfuscated scripts that evade AV signatures.

5.3.4. Comparison with DL-Based Approaches. Our proposed models are evaluated against various DL-based approaches: 1) MalConv2 [17] using byte-level features, 2) MalConv2 with SCORE-T (BFT) features, 3) AST-based classifiers from [11] (SAST, GAST, UAST), and 4) CodeBERT with XGBoost classifier. This comparison demonstrates the benefits of syntactic structure over byte-level representations, effectiveness against other AST/graph-based approaches, and superiority over foundational model representations.

As shown in Table 2, most DL models achieve superior TPR compared to ClamAV and BitDefender with slight FPR increases. However, GAST [11] shows weaker performance in low FPR ranges despite comparable AUROC, since graph convolutional networks capture only spatial correlations between AST nodes, unlike the spatio-temporal analysis in SM-based approaches using LSTMs. MalConv2 excels compared to SAST and UAST due to direct large file processing as raw byte-strings. Performance improves when MalConv2 uses SCORE-T (BFT) features, highlighting SCORE-T efficacy. Similarly, integration with SCORE-T significantly benefits both graph/sequence models. GRL's threat-wise similarity learning achieves remarkable TPR of 0.97855, while GRL's superior performance over CodeBERT-XGBoost shows the richer information provided by graph representations.

SM with SCORE-T (BFT) achieves the highest TPR (0.98088) and optimal FPR, benefiting from hierarchical embeddings and BFT traversal to surpass both GRL approaches, which rank as the second-best performers. These results indicate that SCORE-T features effectively encode syntactic structure and script behavior. When combined with hierarchical or contrastive embedding strategies, they outperform byte-level, AST-only, and foundation-model embeddings, providing robust and accurate malware detection..

5.4. Threat Coverage and Robustness

SM with SCORE-T (BFT) achieves $\geq$90% coverage for most VT threat families and threat types, including scripts missed by VT (see detailed analysis in appendix in Table 4 and Table 5). Robustness evaluation (Table 3) shows accurate detection of obfuscation techniques such as Base64 and XOR, even without decoding, demonstrating model resilience.

TABLE 3: Distribution of obfuscated test samples in the test set and detection accuracy for SM with SCORE-T (BFT).

Obfuscation	Count	Ratio,%	Accuracy,%
Base64 Encoding	3349	44.65	98.38
XOR Encryption	5	0.067	100

6. Related Work

Various studies focus on client-side script languages, particularly JavaScript [26]–[28], while server-side script malware detection remains under-studied in DL contexts. We propose script malware detection for server-side languages using syntactic feature extraction and DL-based techniques as a novel cloud environment mitigation.

Recent DL-based malware detectors convert binaries into grayscale images for computer vision approaches [29]–[33], but these methods are unsuitable for scripts lacking fixed structures, which require contextual code understanding rather than visual pattern analysis.

MalConv [16] uses raw byte sequences as input in a CNN-based model processing over two million steps, though with memory inefficiency. MalConv2 [17] improved memory efficiency and training through temporal max pooling, capturing overall context and detailed features via attention mechanisms. While general-purpose, it struggles with sophisticated malicious behavior and obfuscation evasion.

The UAST approach [11] considers ASTs of programming languages. It combines sequence-based (SAST) and graph-based (GAST) networks, feeding structured AST representations to GCNs and serialized versions to bi-LSTM networks. However, these relatively simple networks (single hidden layers, flattened structures) are insufficient for learning complex malicious behavior.

GraphMatching [25] proposes graph similarity learning for control-flow-graphs, using GNNs to embed graphs for binary function similarity search and vulnerability detection.

CodeBERT [13] captures syntax and meaning of both natural and programming languages using BERT architecture, achieving state-of-the-art results in code intelligence benchmarks. While excelling in textual understanding, AST-based methods can provide deeper syntactic structure modeling crucial for precise functional interpretation.

Our approach targets static malicious behavior detection in scripts, demonstrating significant outperformance over existing methods that either focus on benign code detection or elaborate CFG-based reverse engineering, without considering script malware detection applications.

7. Conclusions

We introduced novel approaches for script malware detection using code representation through ASTs, complemented by SM and GRL. Our experiments show SM with SCORE-T outperforms AVs, benchmark models including existing AST-based models, demonstrating the effectiveness of joint byte-syntactic features. Its enriched hierarchical structure further improves detection rates, with SM excelling when using breadth-first traversal serialization and GRL performing best with threat family labels. The SM processing the hierarchical AST embeddings (BFT) offers the best TPR-FPR trade-off, effectively capturing malicious script behavior.

References

[1] Matt Muir. Legion: an aws credential harvester and smtp hijacker. https://www.cadosecurity.com/legion-an-aws-credential-harvester-and-smtp-hijacker/, 4 2023.

[2] Alex Delamotte. Dissecting alienfox: The cloud spammer's swiss army knife. https://www.sentinelone.com/labs/dissecting-alienfox-the-cloud-spammers-swiss-army-knife/, 3 2023.

[3] Ponemon Institute. The third annual study on the state of endpoint security risk. 1 2020.

[4] Acronis International GmbH. Using ai to detect malicious documents and scripts. 1 2024.

[5] Ravie Lakshmanan. New python-based snake info stealer malware targeting cloud services. https://thehackernews.com/2024/03/new-python-based-snake-info-stealer.html, 3 2024.

[6] Fekun Cyberye Huang and Marshall Chen. Outlaw hacking group's botnet observed spreading miner, perl-based backdoor. https://www.trendmicro.com/en_vn/research/19/f/outlaw-hacking-groups-botnet-observed-spreading-miner-perl-based-backdoor.html, 2 2019.

[7] Richi Jennings. Malicious shell script steals cloud credentials. https://www.trendmicro.com/en_vn/research/21/a/malicious-shell-script-steals-cloud-credentials.html, 3 2021.

[8] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Annual Computer Security Applications Conf. (ACSAC 2007)*, pages 421–430, 2007.

[9] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In *USENIX Security Symposium*, Washington, D.C., August 2003. USENIX Association.

[10] Paras Jain, Ajay Jain, Tianjun Zhang, Pieter Abbeel, Joseph Gonzalez, and Ion Stoica. Contrastive code representation learning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conf. on Empirical Methods in Natural Lang. Proc.*, pages 5954–5971, Online and Punta Cana, Dominican Republic, November 2021. Assoc. for Comput. Linguistics.

[11] Kesu Wang, Meng Yan, He Zhang, and Haibo Hu. Unified abstract syntax tree representation learning for cross-language program classification. In *2022 IEEE/ACM 30th Int, Conf. on Program Comprehension (ICPC)*, pages 390–400, 2022.

[12] Sahil Suneja, Yunhui Zheng, Yufan Zhuang, Jim Laredo, and Alessandro Morari. Learning to map source code to software vulnerability using code-as-a-graph. *ArXiv*, abs/2006.08614, 2020.

[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, November 2020. Association for Computational Linguistics.

[14] Colin Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. Pymt5: multi-mode translation of natural language and python code with transformers. In *2020 Conf. on Empirical Methods in Natural Lang. Proc.*, page 9052–9065, 11 2020.

[15] Chia-Mei Chen, Shi-Hao Wang, Dan-Wei Wen, Gu-Hsin Lai, and Ming-Kung Sun. Applying convolutional neural network for malware detection. In *2019 IEEE 10th Int. Conf. on Awareness Science and Technology (iCAST)*, pages 1–5, 2019.

[16] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware detection by eating a whole EXE. In *The Workshops of the The Thirty-Second AAAI Conf. on AI, New Orleans, Louisiana, USA, February 2-7, 2018*, volume WS-18 of *AAAI Technical Report*, pages 268–276. AAAI Press, 2018.

[17] Edward Raff, William Fleshman, Richard Zak, Hyrum S. Anderson, Bobby Filar, and Mark McLean. Classifying sequences of extreme length with constant memory applied to malware detection. *Proceedings of the AAAI Conf. on AI*, 35(11):9386–9394, 5 2021.

[18] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[19] Yehuda Gelb. Threat actor continues to plague the open-source ecosystem with sophisticated info-stealing malware. https://checkmarx.com/blog/threat-actor-continues-to-plague-the-open-source-ecosystem-with-sophisticated-info-stealing-malware/, 9 2023.

[20] Marc Ohm and Charlene Stuke. Sok: Practical detection of software supply chain attacks. In *Proceedings of the 18th Int. Conf. on Availability, Reliability and Security*, ARES '23, New York, NY, USA, 2023. Association for Computing Machinery.

[21] Duc-Ly Vu, Zachary Newman, and John Speed Meyers. Bad snakes: Understanding and improving python package index malware scanning. In *2023 IEEE/ACM Int. Conf. on Software Eng. (ICSE)*, 5 2023.

[22] GitHub Team. Tree-sitter: a new parsing system for programming tools. https://github.com/tree-sitter/tree-sitter, 2017.

[23] Yuanzhi Ke and Masafumi Hagiwara. Cnn-encoded radical-level representation for japanese processing. *Transactions of the Japanese Society for AI*, 33:D–I23, 07 2018.

[24] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th Int. Conf. on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4189–4198. PMLR, 8 2017.

[25] Li Yujia, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the Int. Conf. on Machine Learning*, pages 3835–3845, 2019.

[26] Doina Cosovan, Razvan Benchea, and Dragos Gavrilut. A practical guide for detecting the java script-based malware using hidden markov models and linear classifiers. In *2014 16th Int. Sym. on Symbolic and Numeric Algorithms for Scientific Computing*, pages 236–243, 2014.

[27] Jack W. Stokes, Rakshit Agrawal, Geoff McDonald, and Matthew Hausknecht. Scriptnet: Neural static analysis for malicious javascript detection. In *MILCOM 2019 - 2019 IEEE Military Comm. Conf. (MILCOM)*, pages 1–8, 2019.

[28] Timon Van Overveldt, Christopher Kruegel, and Giovanni Vigna. Flashdetect: Actionscript 3 malware detection. In Davide Balzarotti, Salvatore J. Stolfo, and Marco Cova, editors, *Research in Attacks, Intrusions, and Defenses*, pages 274–293, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[29] Pratikkumar Prajapati and Mark Stamp. *An Empirical Analysis of Image-Based Learning Techniques for Malware Classification*, pages 411–435. Springer International Publishing, Cham, 2021.

[30] Dipendra Pant and Rabindra Bista. Image-based malware classification using deep convolutional neural network and transfer learning. In *Proceedings of the 3rd Int. Conf. on Advanced Information Science and System*, AISS '21, New York, NY, USA, 2022. Association for Computing Machinery.

[31] Scott Freitas, Rahul Duggal, and Duen Horng Chau. Malnet: A large-scale image database of malicious software. In *Proceedings of the 31st ACM Int. Conf. on Information & Knowledge Management*, CIKM '22, page 3948–3952, New York, NY, USA, 2022. Association for Computing Machinery.

[32] Sang Ni, Quan Qian, and Rui Zhang. Malware identification using visualization images and deep learning. *Comput. Secur.*, 77(C):871–885, 8 2018.

[33] Abien Fred Agarap and Francis John Hill Pepito. Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (svm) for malware classification. *ArXiv*, abs/1801.00318, 2017.

[34] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conf. (ACSAC)*, 12 2020.

Appendix A.
Threat Coverage Analysis

We delve deeper into the threat coverage of our top-performing model, i.e., SM with SCORE-T (BFT). Table 4 show threat family coverage of SM with SCORE-T (BFT) for top 30 threats in the test set. These threats are classified according to VT threat-naming, which incorporates 64 vendors in addition to BitDefender. Threat labeling is performed using AVClass [34], which provides a label based on a consensus among multiple AV vendors. **VT Unknown** represents the malicious samples that were not found in VT database but manually identified as malicious. These scripts are sourced from our HP and they do not exist in VT. **VT Missed** represents the scripts that were labeled as benign by VT but identified as malicious by our model and manual inspection. Singleton also means AVClass [34] cannot identify a family name. Overall, SM with SCORE-T (BFT) achieves >90 % coverage for most families, including those missed by VT, though performance is lower for yellowdye, pyagent, remoteshell, and c99shell. As shown in Table 5, the model attains high coverage for critical categories such as backdoor, cryptominer, webshell, and ransomware, underscoring its effectiveness.

TABLE 4: Threat coverage of SM with SCORE-T (BFT) on test set (VT threat families).

Family	Success	Miss	Ratio%	Family	Success	Miss	Ratio%
singleton	842	15	98.25	surfbuyer	8	0	100
bundlore	791	2	99.75	dstealer	6	0	100
bash	308	0	100	loudminer	6	0	100
meterpreter	108	0	100	shellbot	6	0	100
medusalocker	74	0	100	cimpliads	5	1	83.33
disco	42	1	97.67	rocke	4	0	100
mackeeper	39	1	97.5	necrobot	4	0	100
pykeylogger	34	0	100	tirrip	4	1	80
rozena	33	1	97.06	pypws	4	0	100
shlayer	32	0	100	chopper	4	0	100
mirai	29	0	100	mccrash	4	0	100
memlod	19	0	100	veil	3	0	100
dispread	16	0	100	lazagne	3	0	100
pbot	14	0	100	yellowdye	0	2	0
morila	10	0	100	pyagent	0	2	0
kinsing	9	0	100	remoteshell	0	1	0
discorder	8	0	100	c99shell	0	1	0
VT Unknown	1142	25	97.85	**VT Missed**	79	7	91.86

TABLE 5: Threat coverage of SM with SCORE-T (BFT) on the Test Set (VT threat types).

Threat Type	Success	Miss	Ratio%	Threat Type	Success	Miss	Ratio%
downloader	940	4	99.58	keylogger	11	0	100
grayware	907	6	99.34	grayware:tool	11	0	100
backdoor	146	5	96.69	expkit	10	0	100
worm	101	1	99.02	bot	6	0	100
cryptominer	95	3	96.94	webshell	6	1	85.71
VT Unknown	1895	42	97.83	ransomware	6	1	85.71

Appendix B.
Latency Analysis

Table 6 presents latency performance comparison measured in average milliseconds per file on an Intel Xeon CPU E5-2686 v4 @ 2.30GHz using PyTorch 2. BitDefender and ClamAV emerge as fastest detectors as rule-based AVs, followed by GAST (simple GCN) and MalConv2 (CNN). RNN-based models exhibit 35-50ms latency due to increased architectural complexity. GRL introduces higher latency at 250ms per file due to complex graph propagation, while CodeBERT-XGBoost registers the highest latency at 800ms. Despite latency disparities, DL-based models excel in accuracy compared to BitDefender and ClamAV. This represents the common cybersecurity trade-off between latency efficiency and detection capability. Standard AVs suit tasks prioritizing swift processing of large file volumes and real-time scanning, while SM with SCORE-T, though slower, provides superior accuracy for comprehensive threat detection needs.

TABLE 6: Average latency for the proposed models and benchmarks per file in milliseconds.

Malware detectors	Features	Avg. latency / file
ClamAV	Files	< 1 ms
BitDefender	Files	< 1 ms
[17] MalConv2	Bytes	20 ms
[11] SAST	AST	35 ms
[11] GAST	AST	4 ms
[11] UAST	AST	40 ms
[Ours] SM	SCORE-T	45 ms
[Ours] GRL	SCORE-T	250 ms
CodeBERT-XGBoost	CodeBERT embed.	800 ms

Appendix C.
Programming Language Coverage Analysis

We assess SM with SCORE-T (BFT) across each PL in our training set. Table 7 shows its accuracy for the balanced test set (malicious-benign), the *Rest of Benign* set (comprising 310K benign data not utilized in balanced training, validation or testing), and the *VT benign* set (comprising files from VT with no malicious verdicts). In the test set, we observe lower accuracy for Perl resulting from limited training scripts, though this difference is less obvious in benign sets. Since lack of malicious verdicts from VT doesn't guarantee benign files, the dataset may contain unidentified malicious files unflagged by VT vendors. Therefore, slightly lower performance in the *VT benign* set compared to *Rest of Benign* is expected due to dataset composition uncertainty.

TABLE 7: Script language accuracy of SM with SCORE-T (BFT) on test set, rest of benign and VT benign sets.

Script Language	Test Set	Rest of Benign	VT Benign
Bash	0.99119	0.99933	0.94921
Python	0.97411	0.99952	0.91094
Perl	0.96667	0.99107	1