ShellSight-LLM: Detecting Successful Webshell Intrusions via Optimized LLM

Jiadong Wei[1,2], Jinxia Wei[1,2], Wei Wan[1,2], Hao Fu[1,2], Yuhai Lu[1], Chun Long[1,2(✉)] and Degang Sun[1,2]

[1] Computer Network Information Center, Chinese Academy of Sciences, Beijing 100083, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China
anquanip@cnic.cn

Abstract. Webshells pose a serious threat to network security, as successful webshell intrusions can lead to full compromise of web servers, underscoring the need for accurate and efficient detection methods. Most prior studies focus on detecting webshell intrusions without assessing their success, potentially overlooking the most urgent and damaging incidents. Existing approaches for detecting successful webshell intrusions typically rely on manually selected features to train machine learning models, which suffer from limited generalization and reduced effectiveness against novel intrusion patterns. To address these challenges, this paper proposes ShellSight-LLM, a framework for detecting successful webshell intrusions via an optimized large language model (LLM). By leveraging LLM's powerful semantic understanding and pattern recognizing capabilities, our approach utilizes prompt engineering and in-context learning to effectively capture and convey key characteristics of successful webshell intrusions, thereby enhancing generalization. Furthermore, we employ model distillation to develop a lightweight LLM that retains high detection accuracy while accelerating detection, facilitating practical deployment. Experimental results demonstrate that our method achieves an accuracy of 99.76% and an F1 score of 99.74%, significantly outperforming existing methods while providing improved generalization.

Keywords: Webshell, Intrusion Detection, Network Security, Large Language Model.

1 Introduction

Webshells are malicious scripts implanted on web servers serving as persistent and remote backdoors. They enable attackers to execute commands, upload or modify files, exfiltrate sensitive data, and perform other malicious activities, posing serious threats to network security. According to a report by Cisco Talos Incident Response, webshell deployment attempts against vulnerable web applications were observed in 35% of the incidents in Q4 2024, a sharp increase from less than 10% in the previous quarter [1]. Given the escalating prevalence and destructive potential of webshells, it is crucial to identify successful webshell intrusions accurately.

Previous research on webshell detection can be broadly categorized into two approaches: host-based and traffic-based webshell detection. Host-based methods identify webshell scripts on web server but face several limitations, including the necessity for server-side deployment, reliance on periodic scanning that introduces detection delays, and inability to detect memory-resided webshells. Traffic-based methods monitor network traffic for signs of intrusions, but they typically do not determine whether a webshell intrusion was successful, which can easily generate a large number of low-quality webshell alerts and lead to the neglect of the most urgent and critical alerts. Few studies have focused on traffic-based detection of successful webshell intrusions, and most of these studies rely on manually selected features to train machine learning models, which limits generalization across diverse attack patterns and hinders identification of novel threats. While large language models (LLMs) have shown great potential in network security, they face challenges in this domain due to limited knowledge of webshell specific characteristics. To address these limitations, we propose ShellSight-LLM, a framework for detecting successful webshell intrusions via an optimized LLM. Our approach employs prompt engineering and in-context learning to effectively capture and represent critical features of successful webshell intrusions while leveraging the powerful semantic understanding capabilities of LLMs. Additionally, we incorporate knowledge distillation to enhance detection efficiency and facilitate practical deployment. The proposed solution achieves high detection accuracy and improved generalization for detecting successful webshell intrusions.

The main contributions of this paper are as follows:

1. We present ShellSight-LLM, a LLM based approach for detecting successful webshell intrusions that achieves 99.76% accuracy and 99.74% F1 score, significantly outperforming existing methods.
2. We develop a comprehensive optimization framework to enhance the generalization and performance of LLMs in detecting successful webshell intrusions. Specifically, we employ prompt engineering techniques to capture and convey precise features related to successful webshell intrusions, and provide lexically similar webshell intrusion samples to support effective in-context learning for obfuscated inputs.
3. We introduce a knowledge distillation technique that utilizes prompt-response pairs generated by a teacher LLM to train a lightweight student LLM, which retains high detection accuracy while significantly reducing computational cost, enabling efficient deployment in real-world production environments.

2 Related Work

This section reviews previous work on webshell detection as well as application of LLMs in cyber security respectively. Specifically, existing studies on webshell detection can be categorized into three types based on data sources and detection objectives: host-based detection of webshell scripts, traffic-based detection of webshell intrusions, and traffic-based detection of successful webshell intrusions.

2.1 Host-based detection of webshell scripts

Host-based approaches focus on detecting webshell scripts directly on the compromised web server. Early research in this domain relied on expert chosen features to train a traditional machine learning model for webshell detection [2-7]. Li et al. [8] proposed ShellBreaker, which extracted eight syntactical and semantical features such as the number of global variables from source codes and employed a random forest classifier for detection. With the development of deep learning techniques, more recent studies have leveraged models such as CNN, RNN, LSTM and GRU to improve detection performance [9-15]. Cheng et al. [16] extracted features from AST node sequence of webshells and built a model based on Text-CNN. Liu et al. [17] proposed a model combining bidirectional GRU and attention mechanism for more effective webshell detection. The emergence of large language models such as BERT has provided new tools for webshell detection [18-20]. An et al. [21] utilized CodeBERT as embedding model and Text-CNN as downstream classifier to detect webshells with long text and lexical ambiguity. Despite these advancements, host-based webshell detection methods possess inherent limitations. They necessitate server-side deployment and typically require periodic file scanning on web servers, introducing delays between webshell implantation and detection. Furthermore, webshells that reside solely in memory without any associated files can effectively evade such detection mechanisms.

2.2 Traffic-based detection of webshell intrusions

Traffic-based approaches focus on analyzing network traffic data to detect webshell activity without access to the host system. Some studies relied solely on statistical characteristics of network flows. Le et al. [22] utilized CICFlowMeter to extract 79-dimensional features from traffic packets and employed a deep neural network model to detect webshell intrusions. However, such methods often neglect the semantic information within traffic payloads and are susceptible to network environment fluctuations. Therefore, researchers have increasingly adopted text embedding models to extract richer semantic features from packet payloads for webshell traffic detection [23-28]. Liu et al. [29] proposed two distinct models for webshell traffic detection based on CNN and RNN respectively. Jiang et al. [30] leveraged the BERT model to generate word embeddings and trained a Text-CNN classifier. Despite these improvements, a key limitation of existing traffic-based webshell detection methods is their inability to determine whether the webshell intrusion has actually succeeded. As a result, these approaches can easily generate a large amount of low-quality webshell alerts in real world scenarios, leading to neglect of the most serious threats.

2.3 Traffic-based detection of successful webshell intrusions

Research on traffic-based detection of successful webshell intrusions remains relatively limited. Yang et al. [25] proposed a method that identifies successful webshell intrusions based on response status code equal to 200 and non-empty response bodies,

under the assumption that failed attempts would return error codes such as 302, 404 or empty responses. However, this assumption does not always hold in practice, as many failed webshell intrusions often trigger normal HTML page responses, resulting in a high false positive rate. Guan and Wang et al. [31, 32] extracted statistical features from HTML responses such as the number of tags, scripts, images and the length of stylesheets, and trained a XGBoost-based decision tree to detect successful webshell traffic. Guan et al. [32] proposed a model that extracted character-level features from response bodies and applied bidirectional LSTM with attention mechanism to improve detection accuracy. Overall, however, research on traffic-based detection of successful webshell intrusions is still in its early stages. Most existing studies rely on manually selected features to train machine learning models, which often exhibit limited generalization across diverse attack patterns and reduced effectiveness against novel threats. Further research is needed to develop more robust and adaptable detection methods.

2.4 Large language models for cyber security

The rapid development of large language models (LLMs) has created new opportunities for their application in cyber security, with an increasing number of studies leveraging LLMs to address intrusion detection and threat analysis challenges effectively [33-35]. Lin et al. [36] proposed Encrypted Traffic Bidirectional Encoder Representations from Transformer (ET-BERT), a model that learned generic traffic representations from large-scale unlabeled encrypted traffic through self-supervised pretraining, achieving state-of-the-art performance across multiple encrypted traffic classification tasks. Ali et al. [37] designed HuntGPT, a network intrusion detection system that combined traditional machine learning methods with explainable AI (XAI) framework, effectively enhancing both the operability and interpretability of the intrusion detection system. As for LLM's application in webshell related task, Ma et al. [38] proposed a hybrid prompt algorithm combining chain-of-thought, tree-of-thought and in-context-learning to guide LLMs in generating webshell escape samples. Although these studies highlight the potential of LLMs in cyber security, current LLMs still lack domain specific knowledge necessary for accurately detecting successful webshell intrusions. This limitation underscores the need for effective optimization techniques to transfer webshell specific knowledge to LLMs.

3 Detecting Successful Webshell Intrusions via Optimized LLM

3.1 System Architecture

The overall system architecture is divided into three stages, as shown in Fig. 1. In the first stage, network traffic packets associated with webshell intrusions are processed to extract key fields relevant to detection, including the host, request URL and response body. The response body is then preprocessed and categorized as either plaintext or encrypted/obfuscated content, which are handled separately in the next step. In the second stage, prompts enriched with webshell-specific features are con-

structed and submitted to LLM to determine whether the traffic indicates a successful webshell intrusion. For encrypted or obfuscated response bodies, the system retrieves the most similar samples along with their corresponding labels from a prebuilt database. These samples are incorporated into few-show prompts to help LLM uncover latent patterns through in-context learning. In the final stage, prompt-response pairs generated by the teacher LLM are reviewed and validated by domain experts and subsequently used to distill a smaller, specialized LLM, which retains high detection accuracy while significantly improves efficiency for deployment in real world environment.

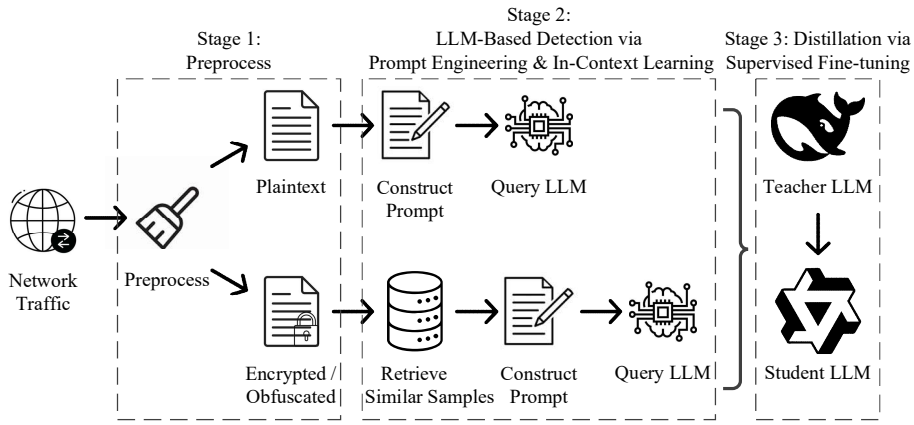

Fig. 1. System architecture of ShellSight-LLM. The framework consists of three stages: (1) preprocessing input traffic and classifying response body as plaintext or encrypted/obfuscated; (2) detecting successful webshell intrusions using LLMs via prompt engineering and in-context learning; (3) distilling a lightweight detection LLM through supervised fine-tuning.

3.2 Preprocessing

The preprocessing stage aims to optimize input for downstream analysis by removing irrelevant information and distinguish between plaintext and encrypted or obfuscated content. It consists of two steps: text cleaning and content classification.

Text cleaning reduces the length of response body while preserving semantic content. For HTML-formatted responses, non-content elements such as styles, scripts and comments are removed to reduce text length without losing meaningful information.

Content classification distinguishes between plaintext and encrypted or obfuscated response bodies. The cleaned text is tokenized using a LLM tokenizer, and the character-to-token ratio r is computed as defined Equation 1, which is then compared against a predetermined threshold ε to determine content type. The rationale behind this method is that plaintext typically yields a higher character-to-token ratio as it contains common words present in tokenizer vocabulary, while encrypted or obfuscated content produces a lower ratio due to its randomness. The threshold is set to $\varepsilon = 1.65$ based on empirical analysis.

$$r = \frac{N_{characters}}{N_{tokens}} \tag{1}$$

3.3 Prompt Engineering

To convey webshell specific features to LLM and enhance model generalization, we adopt a series of prompt engineering techniques to construct tailored prompts for LLM to determine whether the traffic indicates a successful webshell intrusion.

First, we apply iterative prompt refinement through failure case analysis to incorporate precise criteria for identifying successful webshell intrusions. Starting with an initial basic prompt, LLM is queried to evaluate the success of webshell intrusions. We identify discrepancies between LLM's prediction and ground-truth labels, and instruct another LLM to revise the prompt to address these mismatches. Each refinement cycle adds more specific webshell features to the prompt, particularly those associated with previously misclassified samples. This optimization process is repeated over multiple iterations to progressively enhance prompt accuracy and efficiency.

Second, we instruct LLM to generate a reasoning explanation prior to making a final judgement. This encourages the model to perform self-reasoning and ensure its decisions are grounded in logical analysis. Moreover, this approach provides security personnel with transparent insights into the reasoning behind the model's decision-making process.

By incorporating these prompt engineering techniques, we improve both the accuracy and generalization of LLM in detecting successful webshell intrusions. The final optimized prompt is shown in Table 1.

Table 1. Prompt for detecting successful webshell intrusions

Task	You are a cybersecurity expert specializing in intrusion detection. Analyze the provided network packet from a potential webshell invasion attempt and determine whether the webshell invasion is successful or failed based on the response body.
Features of Successful /Failed Webshell Traffic	Indicators of a successful webshell invasion (classify as successful if ANY of the following are present): 1. leaks of sensitive information (e.g., php config, directory listings, log entries, database fields, or other credentials), 2. malicious functionality (e.g., file management, command execution, file upload, email sending, or hash cracking tools), 3. suspicious login page (e.g., simplistic or hidden input form), 4. embedded webshell script, 5. reference to known webshell. Indicators of a failed webshell invasion (classify as failed only if none of the above success indicators are present and any of the following are observed): 1. normal or benign HTML content, 2. error message,

	3. repetition of request URL, indicating the injected command was not executed.
Data	Details to analyze: Host: {host} Request URL: {request_url} Response body: {response_body}
Reasoning &Output	Instructions: Determine whether the webshell invasion attempt is successful or failed based on the response body. Your response should be in the following format: Summary: (Brief summary of the response body) Reason: (One-sentence explanation for your decision) Answer: Successful/Failed

3.4 In-Context Learning

For encrypted or obfuscated response bodies that lack discernible semantic content for LLM to determine successful webshell intrusions, we leverage the in-context learning capabilities of LLM by providing similar samples within the prompt, enabling the model to infer latent patterns from contextual references.

The workflow of in-context learning consists of three parts, as shown in Fig. 2. First, a sample database is constructed by vectorizing the response bodies of training data using an embedding model such as bge-m3 [39]. Then, we encode the response bodies of test data into vectors using the same embedding model, and retrieve the most similar samples from the database based on cosine similarity. The cosine similarity s between two vectors A, B of length n can be calculated as Equation 2. Finally, we compute a lexical text similarity score between test data and retrieved candidates, and rerank these candidates accordingly. The final prompt includes three samples: one with the highest lexical similarity score overall, one with the highest score from the same host, and one with the highest score but an opposite label to the first sample.

$$s = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

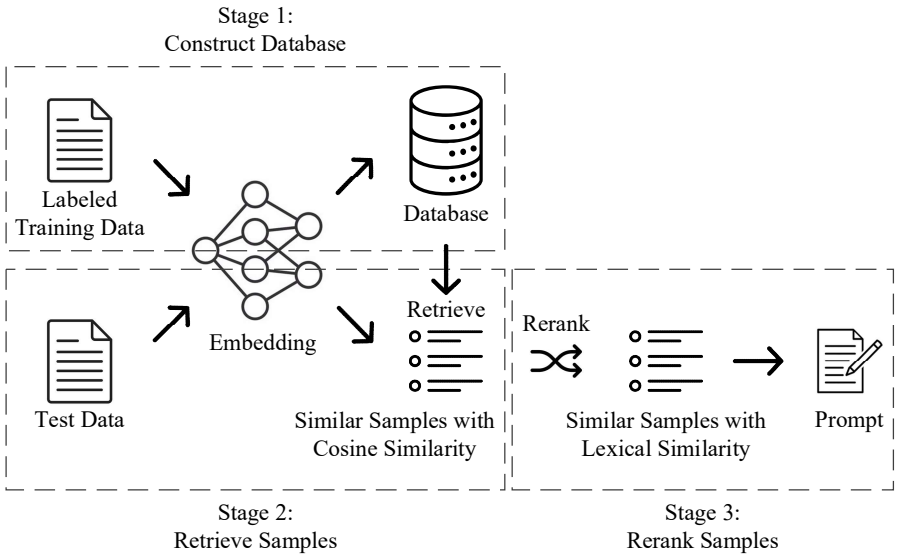

Fig. 2. Workflow of in-context learning. Similar samples are retrieved and reranked for few-shot prompting, enabling the LLM to detect successful webshell intrusions in encrypted or obfuscated responses through in-context learning.

The necessity of reranking and its implementation are described as follows. The embedding model compresses lexical and semantic content of texts into dense vectors, which inevitably results in some loss of information. Therefore, cosine similarity between these vectors may not always accurately reflect the actual similarity between the original texts, potentially yielding suboptimal sample retrievals. To address this limitation, we introduce a lexical text similarity score specifically designed for encrypted webshell traffic and use it to rerank the retrieved candidates for enhanced text similarity. Our approach is motivated by the observation that encrypted webshell responses often exhibit consistent characteristic lexical patterns. For example, Behinder, a widely used webshell tool, generates response bodies that begin with mAUYLzmqn5QPDkyI5lvSp under its default configuration, which decrypts to {"status":"c3VjY2Vzcw==". To capture such lexical features, we employ a 4-gram character model to compute lexical similarity score between test data and each retrieved candidate. The lexical similarity score s between two strings A, B using 4-gram set $4Gram$ is defined in Equation 3. In addition to selecting the most lexically similar sample, we also include samples from the same host as well as samples with an opposite label relative to the first selected sample. This selection strategy ensures that the final prompt contains relevant, diverse and informative samples, thereby improving the effectiveness of in-context learning.

$$s = \left| 4Gram(A) \bigcap 4Gram(B) \right| \tag{3}$$

3.5 Distillation via Supervised Fine-Tuning

Our experiments demonstrate that LLMs with a greater number of parameters exhibit superior performance in detecting successful webshell intrusions, yielding more accurate predictions and coherent explanations. However, these models are also computationally expensive, resulting in slower inference speeds and higher deployment costs. To mitigate these limitations while preserving performance, we employ knowledge distillation via supervised fine-tuning, transferring knowledge from a larger LLM (teacher) to a smaller LLM (student) using prompt-response pairs.

The process of distillation is shown in Fig. 3. First, we collect responses from teacher LLM using prompts for detecting successful webshell intrusions described in Section 3.3 and 3.4. These responses are then reviewed and refined by network security experts to correct any inaccuracies in predictions or explanations. The validated prompt-response pairs are subsequently used to fine-tune student LLM using Low-Rank Adaptation (LoRA) [40]. LoRA is an efficient and effective fine-tuning technique that decompose weight updates Δw into two low-rank metrics *A, B* as defined in Equation 4. Through this process, we obtain an optimized LLM that retains strong detection capabilities while significantly reducing computational costs.

$$\Delta w = BA, B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}, r \ll \min(d, k) \tag{4}$$

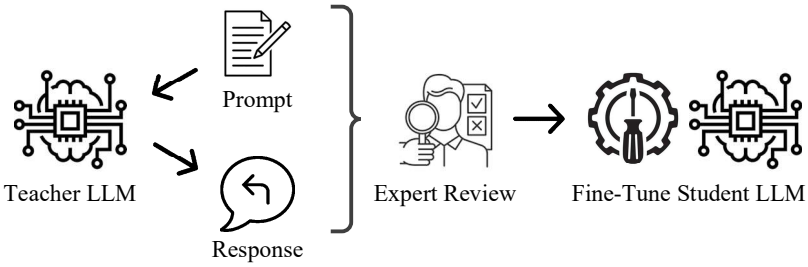

Fig. 3. LLM distillation via supervised fine-tuning. The distillation process transfers knowledge for detecting successful webshell intrusions from a larger LLM (teacher) to a smaller LLM (student) using expert-verified prompt-response pairs.

4 Experiment and Results

4.1 Setup

Experimental settings. We employ DeepSeek-V3 as the teacher model and Qwen2.5-7B as the student model, with responses obtained through API queries. To ensure reproducibility and robustness, the temperature is set to 0.01, minimizing randomness in model outputs. LoRA fine-tuning is performed using LlamaFactory [41] with a LoRA rank of 32 and a learning rate of 0.0002. All experiments are conducted on an Ubuntu server configured with a 2.3GHz Intel Xeon Gold 5218 CPU, an NVIDIA RTX 4090 GPU and 128GB memory.

Datasets. Network traffic data is collected from China Science and Technology Network (CSTNET) over a 3-months period. Utilizing established traffic-based webshell detection techniques, we identify over 1.8 million webshell intrusions. Among these, 224 flows are confirmed as successful webshell intrusions through manual verification, while 11,214 flows classified as failed webshell intrusions are randomly selected. To mitigate the scarcity of successful webshell intrusions in real world network environments, we supplement the dataset by simulating successful webshell intrusions. Various types of webshell scripts are collected from GitHub and deployed on a server in a sandbox environment to simulate compromised systems. We then use browsers and widely-used webshell tools such as Chopper, AntSword, Behinder and Godzilla to interact with these webshells while capturing network traffic using Wireshark, generating 9,041 flows of successful webshell intrusions. Combined with the previously identified real-world flows, the final dataset comprises 9,265 successful and 11,214 failed webshell intrusion traffic flows. This dataset is evenly split into training and testing subsets.

Evaluation metrics. To evaluate and compare the performance of different methods, we adopt 5 commonly used evaluation metrics: accuracy, precision, recall, false positive rate (FPR) and F1 score.

4.2 Comparative Experiment

The comparative results of various methods for detecting successful webshell intrusions are presented in in Table 2.

Table 2. Comparison results of ShellSight-LLM with other methods for detecting successful webshell intrusions.

Method	Accuracy (%)	Precision (%)	Recall (%)	FPR (%)	F1 Score (%)
Response status code	91.83	85.05	99.40	14.43	91.67
XGBoost	95.27	94.91	94.62	4.19	94.77
Bidirectional LSTM	95.78	97.64	92.92	1.85	95.22
BERT+TextCNN	98.23	98.24	97.84	1.44	98.04
ShellSight-LLM (ours)	**99.76**	**99.61**	**99.87**	**0.33**	**99.74**

The baseline methods listed in Table 2 have been discussed in detail in Section 2. They represent a range of approaches, from rule-based techniques such as response status code analysis [25], to statistical learning models such as XGBoost [31, 32], deep learning architectures such as Bidirectional LSTM [32], and pretrained language models such as BERT+TextCNN [30].

As shown in Table 2, our proposed method ShellSight-LLM achieves the best performance across all evaluation metrics, including the highest accuracy, precision, recall, and F1 score, as well as the lowest FPR. This superior performance can be attributed to our carefully designed framework. Specifically, we incorporate explicit characteristics of successful webshell intrusions through prompt engineering, enhance pattern inference via in-context learning with diverse and representative examples, and further refine the model through knowledge distillation using supervised fine-

tuning. These techniques, combined with the powerful semantic understanding capabilities of LLM, contribute to ShellSight-LLM's superior effectiveness.

The BERT+TextCNN method ranks second in overall performance, demonstrating the effectiveness of pretrained language models in this task. However, its decision-making process lacks transparency, making it difficult for security experts to analyze misclassifications and refine the model accordingly. In contrast, the response status code method, though simple and computationally efficient, suffers from low precision and high FPR, potentially leading to numerous low-quality alerts and hinders identification of critical threats.

We present several representative examples generated by ShellSight-LLM in Table 3. These examples demonstrate that our method not only achieves accurate classification of successful webshell intrusions but also provides interpretable reasoning to support its decisions.

Table 3. Example prompt-response pairs generated by ShellSight-LLM

Prompt (excerpt)	Response
# … Prompt Templates … Response body: 163089ef7nls/ 2022-10-05 21:17:00 111 RWX dojo.js 2022-10-05 21:17:00 297668 RW- dojo.js.uncompressed.js 2022-10-05 21:17:00 1226798 RW- dojo.js.compressed.js 2024-03-28 12:36:39 4004 RW- e1f164da84	Summary: The response body lists files with their timestamps, permissions, and sizes, including references to JavaScript files like "dojo.js" and its variants. Reason: The presence of file listings and timestamps indicates a directory traversal or file management capability, which is a sign of malicious functionality. Answer: Successful
# … Prompt Templates … Response body: Distributor ID: Debian Description: Debian GNU/Linux 12 (bookworm) Release: 12 Codename: bookworm	Summary: The response body contains the output of the `lsb_release -a` command, which reveals system information about the Debian Linux distribution. Reason: The response leaks sensitive system information (Distributor ID, Description, Release, and Codename), indicating successful command execution by the webshell. Answer: Successful
# … Prompt Templates … Response body: {"timestamp":1708767117867, "status":404, "error":"Not Found", "message":"No message available", "path":"/_phpMyAdmin/scripts/setup.php"}	Summary: The response body indicates a 404 error with the message "Not Found" and "No message available" for the requested path "/_phpMyAdmin/scripts/setup.php". Reason: The response shows an error message and no evidence of successful webshell indicators. Answer: Failed

4.3 Ablation Experiment

To evaluate the contribution of various components such as prompt engineering, in-context learning, reranking and distillation to the overall performance of our method, we conduct ablation experiments by selectively removing each component. The results are summarized in Table 4.

Table 4. Comparison results of ablation experiment

Model	Accuracy (%)	Precision (%)	Recall (%)	FPR (%)	F1 Score (%)
ShellSight-LLM (ours)	99.76	99.61	99.87	0.33	99.74
w/o prompt engineering	98.03	99.07	96.60	0.76	97.82
w/o in-context learning	96.89	95.11	98.19	4.17	96.62
non-selective in-context learning	99.14	98.20	99.91	1.48	99.05
w/o reranking	98.78	98.86	98.42	0.93	98.64
Qwen2.5-7B w/o fine-tuning	97.40	99.37	94.85	0.50	97.06
DeepSeek-V3 w/o fine-tuning	99.61	99.63	99.50	0.30	99.57

In w/o prompt engineering, we replace our optimized prompt with a basic instruction for detecting successful webshell intrusions. This leads to a notable drop in recall (-3.27), suggesting that well-designed prompt is essential for aligning model's criteria for successful webshell intrusions with human expert. In w/o in-context learning, the model receives no relevant examples. This particularly hampers performance on encrypted or obfuscated response bodies, resulting in significant decreases in precision (-4.50). The non-selective in-context-learning setting provides the model with contextual examples even for plaintext cases. However, performance does not improve further and slightly drops (precision -1.41), indicating that redundant or unnecessary examples may introduce noise and obscure critical features. In w/o reranking, examples are selected purely based on cosine similarity without lexical reranking. The performance degradation (accuracy -0.98) confirms the effectiveness of reranking for selecting the most relevant samples, which in turn enhances model effectiveness. Finally, we evaluate the performance of Qwen2.5-7B and DeepSeek-V3 without distillation. Our distilled version of Qwen2.5-7B significantly outperforms its original version (accuracy +2.36) and closely matches, or even slightly surpasses, the teacher model DeepSeek-V3 (accuracy +0.15). This demonstrates the effectiveness of expert-guided distillation in transferring knowledge and enhancing model performance.

4.4 Generalization Experiment

To evaluate and compare the generalization capabilities of different methods, we adopt a more challenging train-test split strategy based on host and URL. Specifically, network traffic that shares the same host and URL is assigned to the same subset, either training or testing, to better simulate real-world scenarios where new, unseen

inputs are common. The results along with the relative performance changes compared to the original split are summarized in Table 5.

Table 5. Comparison results of generalization experiment (Δ indicates change from original split).

Method	Accuracy (%)	Precision (%)	Recall (%)	FPR (%)	F1 Score (%)
XGBoost	92.63 (-2.64)	91.97 (-2.94)	91.76 (-2.86)	6.63 (+2.44)	91.87 (-2.90)
Bidirectional LSTM	90.31 (-5.47)	94.82 (-2.82)	83.15 (-9.77)	3.76 (+1.91)	88.60 (-6.62)
BERT+TextCNN	94.35 (-3.88)	95.52 (-2.72)	91.84 (-6.00)	3.57 (+2.13)	93.65 (-4.39)
ShellSight-LLM (ours)	**99.40 (-0.36)**	**99.39 (-0.22)**	**99.31 (-0.56)**	**0.51 (+0.18)**	**99.35 (-0.39)**

As shown in Table 5, our proposed method, ShellSight-LLM, achieves the best generalization performance with only minimal declines across all evaluation metrics. This robust generalization is largely attributed to the use of large-scale pretrained language models, which excel at capturing abstract patterns and deep semantic representations. Furthermore, our carefully designed prompts emphasize essential features, enabling the model to focus on relevant decision signals even in unfamiliar cases. The results indicate that our method ShellSight-LLM is more capable of handling previously unseen real-world webshell attack scenarios compared to existing approaches, thereby enhancing its effectiveness and reliability in practical deployment.

5 Conclusion

In this paper, we propose ShellSight-LLM, an optimized LLM-based framework for detecting successful webshell intrusions. Our approach achieves 99.76% accuracy and 99.74% F1score, outperforming existing methods. To adapt LLM to this task and enhance generalization, the framework incorporates precise webshell specific features through iterative prompt refinement, and leverages in-context learning by providing similar samples to effectively capture and convey latent intrusion patterns. Furthermore, we employ knowledge distillation to train a lightweight student LLM that retains high detection performance while reducing computational costs, enabling efficient real-world deployment. Experimental results demonstrate that ShellSight-LLM not only surpasses traditional and deep learning baselines in detection accuracy but also exhibits superior generalization ability. However, this study has certain limitations. The prompt-response pairs used for distillation may lack sufficient diversity to fully exploit the model's potential. In future research, we plan to optimize the distillation dataset by improving its representativeness or selectively curating typical examples to facilitate more efficient and effective fine-tuning.

Acknowledgements

This work is supported by Youth Innovation Promotion Association of Chinese Academy of Sciences (No. 2022170).

References

1. Talos IR trends Q4 2024: Web shell usage and exploitation of public-facing applications spike, https://blog.talosintelligence.com/talos-ir-trends-q4-2024/, last accessed 2025/5/23.
2. Fang, Y., Qiu, Y., Liu, L., et al.: Detecting Webshell Based on Random Forest with FastText. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 52-56. Association for Computing Machinery, Chengdu, China (2018).
3. Pan, Z.L., Chen, Y.C., Chen, Y., et al.: Webshell Detection Based on Executable Data Characteristics of PHP Code. Wireless Communications & Mobile Computing 2021(1), 5533963 (2021).
4. Guan, T., Zhang, J., and Mao, J.: Research on Webshell Detection Method Based on Machine Learning. In: 2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE), pp. 1391-1394. IEEE, Xiamen, China (2019).
5. Yong, B.B., Wei, W., Li, K.C., et al.: Ensemble machine learning approaches for webshell detection in Internet of things environments. Transactions on Emerging Telecommunications Technologies 33(6), e4085 (2022).
6. Zhang, H., Liu, M., Yue, Z., et al.: A PHP and JSP Web Shell Detection System with Text Processing Based on Machine Learning. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1584-1591. IEEE, Guangzhou, China (2020).
7. Huang, W., Jia, C., Yu, M., et al.: UTANSA: Static Approach for Multi-Language Malicious Web Scripts Detection. In: 2021 IEEE Symposium on Computers and Communications (ISCC), pp. 1-7. IEEE, Athens, Greece (2021).
8. Li, Y., Huang, J., Ikusan, A., et al.: ShellBreaker: Automatically detecting PHP-based malicious web shells. Computers & Security 87, 101595 (2019).
9. Lu, J., Tang, Z., Mao, J., et al.: Mixed-Models Method Based on Machine Learning in Detecting WebShell Attack. In: Proceedings of the 2020 International Conference on Computers, Information Processing and Advanced Education, pp. 251-259. Association for Computing Machinery, Ottawa, ON, Canada (2020).
10. Le, H., Nguyen, T.N., Nguyen, H.N., et al.: An Efficient Hybrid Webshell Detection Method for Webserver of Marine Transportation Systems. Ieee Transactions on Intelligent Transportation Systems 24(2), 2630-2642 (2023).
11. Li, T.T., Ren, C.H., Fu, Y.S., et al.: Webshell Detection Based on the Word Attention Mechanism. Ieee Access 7, 185140-185147 (2019).
12. Lv, Z., Yan, H., and Mei, R.: Automatic and Accurate Detection of Webshell Based on Convolutional Neural Network. In: Cyber Security, pp. 73-85. Springer, Beijing, China (2019).
13. Nguyen, N.H., Le, V.H., Phung, V.O., et al.: Toward a Deep Learning Approach for Detecting PHP Webshell. In: Proceedings of the 10th International Symposium on Information and Communication Technology, pp. 514-521. Association for Computing Machinery, Hanoi, Ha Long Bay, Viet Nam (2019).

14. Zhang, Z., Li, M., Zhu, L., et al.: SmartDetect: A Smart Detection Scheme for Malicious Web Shell Codes via Ensemble Learning. In: Smart Computing and Communication, pp. 196-205. Springer Cham, Tokyo, Japan (2018).
15. Zhou, Z., Li, L., and Zhao, X.: Webshell Detection Technology Based on Deep Learning. In: 2021 7th IEEE Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), pp. 52-56. IEEE, New York, USA (2021).
16. Cheng, K., Wang, H., Hu, G., et al.: Research on Webshell Detection Based on Semantic Analysis and Text-CNN. In: 2021 17th International Conference on Computational Intelligence and Security (CIS), pp. 529-534. IEEE, Chengdu, China (2021).
17. Liu, Z.Q., Li, D.F., and Wei, L.L.: A New Method for WebShell Detection Based on Bidirectional GRU and Attention Mechanism. Security and Communication Networks 2022, 11 (2022).
18. Cheng, B., Guo, Y., Ren, Y., et al.: MSDetector: A Static PHP Webshell Detection System Based on Deep-Learning. In: Theoretical Aspects of Software Engineering, pp. 155-172. Springer International Publishing, Cluj-Napoca, Romania (2022).
19. Pu, A., Feng, X., Zhang, Y.H., et al.: BERT-Embedding-Based JSP Webshell Detection on Bytecode Level Using XGBoost. Security and Communication Networks 2022(1), 4315829 (2022).
20. Wang, G., Ko, H., Chiang, C., et al.: WebShell Detection Based on CodeBERT and Deep Learning Model. In: Proceedings of the 2024 5th International Conference on Computing, Networks and Internet of Things, pp. 484-489. Association for Computing Machinery, Tokyo, Japan (2024).
21. An, T., Shui, X., and Gao, H.: Deep Learning Based Webshell Detection Coping with Long Text and Lexical Ambiguity. In: Information and Communications Security, pp. 438-457. Springer, Canterbury, UK (2022).
22. Le, H.V., Vo, H.V., Nguyen, T.N., et al.: Towards a Webshell Detection Approach Using Rule-Based and Deep HTTP Traffic Analysis. In: Computational Collective Intelligence, pp. 571-584. Springer International Publishing, Hammamet, Tunisia (2022).
23. Tao, F., Cao, C., and Liu, Z.: Webshell Detection Model Based on Deep Learning. In: Artificial Intelligence and Security, pp. 408-420. Springer International Publishing, New York, USA (2019).
24. Tian, Y., Wang, J., Zhou, Z., et al.: CNN-Webshell: Malicious Web Shell Detection with Convolutional Neural Network. In: Proceedings of the 2017 VI International Conference on Network, Communication and Computing, pp. 75-79. Association for Computing Machinery, Kunming, China (2017).
25. Yang, W., Sun, B., and Cui, B.: A Webshell Detection Technology Based on HTTP Traffic Analysis. In: Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 336-342. Springer International Publishing, Matsue, Japan (2019).
26. Zhang, H., Guan, H.C., Yan, H.B., et al.: Webshell Traffic Detection With Character-Level Features Based on Deep Learning. Ieee Access 6, 75268-75277 (2018).
27. Wu, Y.X., Sun, Y.Q., Huang, C., et al.: Session-Based Webshell Detection Using Machine Learning in Web Logs. Security and Communication Networks 2019(1), 3093809 (2019).
28. Wu, Y., Song, M., Li, Y., et al.: Improving Convolutional Neural Network-Based Webshell Detection Through Reinforcement Learning. In: Information and Communications Security, pp. 368-383. Springer International Publishing, Chongqing, China (2021).

29. Liu, H.Y., Lang, B., Liu, M., et al.: CNN and RNN based payload classification methods for attack detection. Knowledge-Based Systems 163, 332-341 (2019).
30. Jiang, K., Yu, Z., Chen, X., et al.: Multidimensional Webshell Detection Method Based on Deep Learning. In: 2022 15th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), pp. 1-6. IEEE, Beijing, China (2022).
31. Wang, S.: WebShell detection and Research based on HTTP protocol. Beijing University of Posts and Telecommunications, (2021).
32. Guan, H.: Research of WebShell Detection Based on HTTP Traffic. Beijing University of Posts and Telecommunications, (2019).
33. Ferrag, M.A., Ndhlovu, M., Tihanyi, N., et al.: Revolutionizing Cyber Threat Detection With Large Language Models: A Privacy-Preserving BERT-Based Lightweight Model for IoT/IIoT Devices. Ieee Access 12, 23733-23750 (2024).
34. Yu, J., Choi, Y., Koo, K., et al.: A novel approach for application classification with encrypted traffic using BERT and packet headers. Computer Networks 254, 110747 (2024).
35. Wang, Z., Li, J., Yang, S., et al.: A lightweight IoT intrusion detection model based on improved BERT-of-Theseus. Expert Systems with Applications 238, 122045 (2024).
36. Lin, X., Xiong, G., Gou, G., et al.: ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification. In: Proceedings of the ACM Web Conference 2022, pp. 633–642. Association for Computing Machinery, Virtual Event, Lyon, France (2022).
37. Ali, T. and Kostakos, P.: HuntGPT: Integrating Machine Learning-Based Anomaly Detection and Explainable AI with Large Language Models (LLMs). ArXiv 2309.16021 (2023).
38. Ma, M., Han, L., and Zhou, C.: Large Language Models are Few-shot Generators: Proposing Hybrid Prompt Algorithm To Generate Webshell Escape Samples. ArXiv 2402.07408 (2024).
39. Chen, J., Xiao, S., Zhang, P., et al.: M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In: Findings of the Association for Computational Linguistics: ACL 2024, pp. 2318-2335. Association for Computational Linguistics, Bangkok, Thailand (2024).
40. Hu, E.J., Shen, Y., Wallis, P., et al.: Lora: Low-rank adaptation of large language models. In: The International Conference on Learning Representations (ICLR), pp. 3. ICLR, Virtual (2022).
41. Zheng, Y., Zhang, R., Zhang, J., et al.: LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, pp. 400-410. Association for Computational Linguistics, Bangkok, Thailand (2024).