Large Language Models are Few-shot Generators: Proposing Hybrid Prompt Algorithm To Generate Webshell Escape Samples

Anonymous ACL submission

Abstract

The frequent occurrence of cyber-attacks has made webshell attacks and defense gradually become a research hotspot in the field of network security. However, the lack of publicly available benchmark datasets and the over-reliance on manually defined rules for webshell escape sample generation have slowed down the progress of research related to webshell escape sample generation and artificial intelligence-based webshell detection. To address the drawbacks of weak webshell sample escape capabilities, the lack of webshell datasets with complex malicious features, and to promote the development of webshell detection, we propose the Hybrid Prompt algorithm for webshell escape sample generation with the help of large language models. As a prompt algorithm specifically developed for webshell sample generation, the Hybrid Prompt algorithm not only combines various prompt ideas including Chain of Thought, Tree of Thought, but also incorporates various components such as webshell hierarchical module and few-shot example to facilitate the LLM in learning and reasoning webshell escape strategies. Experimental results show that the Hybrid Prompt algorithm can work with multiple LLMs with excellent code reasoning ability to generate high-quality webshell samples with high Escape Rate (88.61% with GPT-4 model on VIRUSTOTAL detection engine) and Survival Rate (54.98% with GPT-4 model).

1 Introduction

Webshell (Starov et al., 2016), as a typical example of malicious scripts exploiting injection vulnerabilities, allows hackers to remotely access and invade web servers, posing serious threats to socio-economic and network security. Webshells come in various forms, ranging from a single line of code that allows remote execution of user-provided system commands to large-scale complex script files. Similar to the research on malware detection, webshell generation and detection are non-stationary, adversarial problems (Demetrio et al., 2021), which have been engaged in a constant game of cat and mouse, with an escalating spiral trend. From the attacker's perspective, mainstream webshell detection tools and engines like VIRUSTOTAL (Peng et al., 2019), WEBDIR+ and AVAST are frequently updated and maintained, incorporating the rules and characteristics of new webshells within days or even shorter periods. This forces attackers to constantly develop new webshell generation methods to bypass the detection of such engines. On the detection side, research is still in its infancy (Hannousse and Yahiouche, 2021). There is a lack of publicly available benchmark datasets and open-source baseline methods for webshell detection. Most models using neural networks or intelligent algorithms claim to have high accuracy and low false positives. However, the fact is that these models are basically tested on private datasets, which usually consist of only a few hundred or fewer samples, with obvious malicious features. Even the simplest multi-layer perceptron (MLP) structure can achieve high-precision detection on such datasets through overfitting. In a real cyber-attack environment, the authenticity and generalization ability of such methods are difficult to guarantee. See details in App. A.

In fact, (Hannousse and Yahiouche, 2021) argued that AI methods excel at extracting abstract features in webshell, which are advanced features that go beyond lexical, syntactical, and semantical features. These advanced features help reveal hidden aspects in webshells that cannot be detected through syntax and semantic analysis. However, unlike the research on malware adversarial sample generation (Demetrio et al., 2021; Kolosnjaji et al., 2018; Song et al., 2022; Castro et al., 2019), research on webshell escape sample generation is still a blank field, which is due to the fact that the

existing webshell bypass strategies are numerous and complicated, and there is no specific systematic method to follow. Therefore, it is an urgent and highly significant work to propose a webshell escape sample generation algorithm and construct a corresponding webshell benchmark dataset.

On the other hand, the blooming development of large language model (LLM) and artificial intelligence generated content (AIGC) technologies (Chang et al., 2023) has already made an indelible impact in various domains such as chat and image generation (Zhao et al., 2023b). As the latest achievement in the field of natural language processing (NLP), LLM has taken a significant lead over earlier neural network structures (i.e. Long Short-Term Memory (Staudemeyer and Morris, 2019), Gate Recurrent Unit (Dey and Salem, 2017), etc.) in contextual reasoning and semantic understanding capabilities. The widespread application of LLM in various code-related tasks (i.e. code generation (Liu et al., 2023a), penetration testing (Deng et al., 2023), vulnerability detection (Sun et al., 2023), automated program repair (Wei et al., 2023), LLM fuzz tuning (Zhao et al., 2023a), vulnerability repair (Pearce et al., 2023)) has fully showcased its excellent code reasoning abilities, making it possible to utilize LLM for generating webshell escape samples. Prompt engineering (Shin et al., 2020) plays a crucial role in the vertical research application of LLM, which aims to explore better ways of human interaction with LLMs to fully leverage their performance potential. It is undeniable that many key techniques in prompt engineering, such as Chain of Thoughts (CoT) (Wei et al., 2022), Tree of Thoughts (ToT) (Yao et al., 2023), Zero-Shot CoT (Kojima et al., 2022), etc., have improved the reasoning abilities of LLMs. In addition, the application of techniques like LAnguage Model Analysis (LAMA) probes (Petroni et al., 2019) have been gradually enhancing the interpretability of the models. Novel studies in prompt engineering, such as prompt finetuning, have been able to fine-tune the parameters in LLM, thus simplifying the traditional fine-tune process (Li and Liang, 2021). Moreover, AIGC technology is so "creative", that just a simple prompt can make LLM produce a 0-Day webshell, see details in App. B.

Therefore, in this work, we explore the unexplored research area of AIGC-enabled webshell escape sample generation strategies. We propose Hybrid Prompt, a hierarchical and modular prompt generation algorithm, and apply it to different LLM models to generate multiple webshell samples with high escape capabilities. Experimental results demonstrate that the escape samples generated by the Hybrid Prompt algorithm + LLM model can bypass detection by mainstream detection engines with high Escape Rate (ER) and Survival Rate (SR).

The main contributions of this paper are threefolds:

- We propose Hybrid Prompt algorithm, which combines the advantages of multiple prompt schemes such as ToT (Yao et al., 2023), few-shot prompting, CoT, etc. By synthesizing key features related to webshell escape and designing prompt strategies tailored to different sizes of webshells, the algorithm effectively enhances the code reasoning ability of LLM models and generates high-quality webshell escape samples.

- We construct a webshell benchmark dataset generated by the Hybrid Prompt algorithm. This dataset achieves high ER and SR among mainstream detection engines and reflects the performance of rule-based detection engines more realistically and effectively.

- We investigate and compare the quality of escape samples generated by different LLM models using the Hybrid Prompt algorithm. All these samples exhibit high ER, surpassing webshell samples generated by other intelligent algorithms (i.e. genetic algorithm (Pang et al., 2023)).

2 Preliminary

With the development of AIGC and LLM technologies, there are numerous LLM models in different subfields with different focuses. For example, GLM (Du et al., 2022) and GLM2 models tend to prioritize open-source and lightweight to meet the deployment needs of personal terminals. DALLE (Ramesh et al., 2021) focuses on AI image generation, while FATE-LLM (Zhuang et al., 2023) is biased towards application scenarios under the federal learning paradigm. Hybrid Prompt performs exceptionally well on LLM models with strong code reasoning abilities. We have done some toy tests with basic prompts on Chatglm-6B

[1], Chatglm2-6B [2], Chatglm-13B, and Chatglm2-13B models, but the performance is unsatisfactory, see details in App. C. Therefore, this strategy is more suitable for LLM models with a large number of parameters and strong code reasoning abilities, such as GPT-3.5 and GPT-4.

3 Algorithm Design

3.1 Overall Workflow

The overall flow from collecting multi-source webshell scripts to generating webshell escape samples is shown in Figure 1.

Figure 1: The overall workflow of webshell escape sample generation

3.2 Data Filtering

To facilitate the implementation of the Hybrid Prompt algorithm, we need to construct the Template webshell dataset. Since webshell scripts collected from multiple sources are diverse in types and have confusing names (i.e. AK-74, b374k, etc.), and the Template webshell dataset requires clean and well-characterized webshell scripts, we perform triple data filtering process on multi-source webshell scripts, as shown in Figure 2.

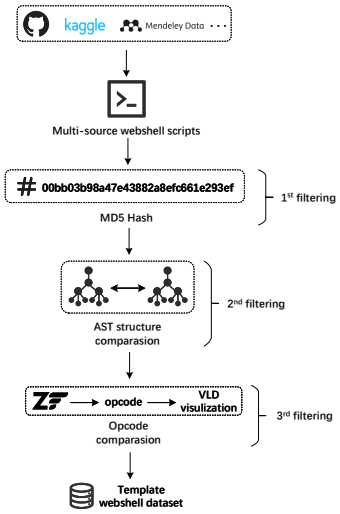

Figure 2: Triple data filtering process

In the first filtering step, we calculate the *MD5 hash* value of all scripts to filter out webshell scripts with consistent content but confusing names. The filtered scripts are then renamed using their corresponding hash values. In the second filtering step, we convert the webshell scripts into Abstract Syntax Tree (AST) structures to filter out the scripts with the same syntax structure. For PHP scripts, we use "*php-ast*" to perform the translation (*ast\parse_code*) and add the *name, kind* attribute to the nodes. We process the child nodes belonging to the array and AST separately. The pseudocode for this step is shown in Algorithm 1.

Algorithm 1 Php-ast Runtime Flow

```
$ast = ast\parse_code($code, $version=70);
$new_ast = add_attr($ast);
$json = json_encode($new_ast, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE | JSON_OBJECT_AS _ARRAY);
```

In the third filtering step, the Vulcan Logic Disassembler (VLD) module in Zend engine is used to disassemble the scripts into opcode structures, aiming to filter out webshell scripts with consistent execution sequences. See details in App. D.

3.3 Hybrid Prompt

The ToT method has significant performance advantages over CoT, Self Consistency (SC) (Wang et al., 2023) method in solving complex reasoning problems by searching for multiple solution paths, using strategies such as backtracking and pruning, similar to human thinking rather than the traditional auto-regressive mechanism of making token level decisions one by one in a left-to-right manner. This allows it to better handle heuristic problems like genuine problems. Therefore, we also leverage and innovate this process paradigm in the complex reasoning task of webshell escape sample generation. The overall flowchart of the Hybrid Prompt algorithm is illustrated in Figure 3.

Before proceeding, let's first formalize some relevant symbols. We use M to denote LLM, o to denote one of the candidates generated by each thought of Hybrid Prompt, O to denote the set composed of candidates, x to denote the original input of Hybrid Prompt, F_e to denote the few-shot example, N to denote the tree depth of Hybrid Prompt and p to denote the number of candidates.

3.3.1 Thought Decomposition

ToT argues that a suitable thought should be able to generate promising and diverse samples, facilitating LLM in assessing its problem-solving prospects. However, compared to tasks with clear

[1] https://github.com/THUDM/ChatGLM-6B
[2] https://github.com/THUDM/ChatGLM2-6B

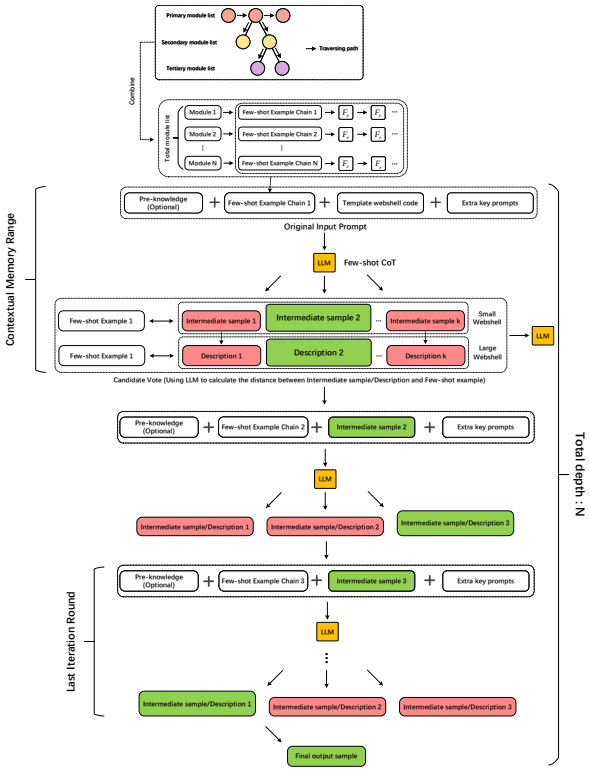

Figure 3: The flowchart of Hybrid Prompt algorithm

rules such as *Game of 24*, *5*5 Crosswords*, etc., the thought search space for webshell escape sample generation is broader and more challenging.

To address this, we have developed a webshell escape sample generation whitepaper by taking into account the characteristics of webshell escape samples. We refer to each keyword as a module, and some modules further have secondary and tertiary modules, see details in App. E. Therefore, in Hybrid Prompt, thought is set as the search space for LLM contemplating Template webshells based on a module.

3.3.2 Thought Generator $G(M, o)$

Since webshell escape sample generation is a heuristic problem, we apply the CoT method to each module for generating multiple intermediate webshell samples. Considering that LLM may generate some low-value solutions with large deviations from the expectation, thus reducing the efficiency of subsequent votes, we design F_e chain structure for each module, see details in App. F.

Therefore, $G(M, o) = M(F_e, o)$.

Each node in the F_e chain includes the original webshell sample, as well as the webshell sample processed by the corresponding module, and a brief description explaining the processing method and core ideas of the module. When filtering the F_e chain, we follow the following 2 principles: 1) The structure of the example webshell code should be as simple as possible; 2) Each node contains, as far as possible, only the processing methods corresponding to that module.

The purpose is to reduce the difficulty of LLM in learning the corresponding method through an example that is as simple as possible and contains the core idea. The descriptive explanation further enhances the interpretability of the solutions. This idea is also in line with the logical process of human learning and cognition, e.g., "from shallow to deep," to help LLM better learn the features of the methods.

F_e can essentially "modify" the LLM's thinking direction to a certain extent so that webshell can be generated in a Few-shot CoT mindset. In most cases, each F_e chain contains multiple F_e examples to provide more comprehensive coverage of different scenarios. In this case, multiple nodes are used as input prompt components for the current iteration round, to help LLM better learn multiple segmented strategies. Due to the large search space and sample diversity for each module, this Few-shot CoT method yields better results.

Meanwhile, based on the input webshell size, we design 2 different generation approaches. For small webshells, we include p candidate webshell samples in a single conversation returned by the LLM. In this case, the average maximum length of each candidate webshell sample $L(Avg\_Candidate_i)$ is calculated as (1):

$$L(Avg\_Candidate_i) = (L(MaxToken) - L(InputPrompt))/p. \quad (1)$$

Where $L(MaxToken)$ denotes the maximum context length that the current LLM model can handle, and $L(InputPrompt)$ denotes the length of the input prompt in the current thought. Since small webshells are generally shorter, this approach can save the consumption of LLM's token resources, and enable LLM to generate more diverse samples in the returned message of a single conversation through specific "key prompts".

For large webshells, we enable the n parameter function to generate p candidate webshell samples by receiving multiple return messages from LLM. In this case, the maximum length of each candidate webshell sample $L(Candidate_i)$ is calculated as (2):

$$L(Candidate_i) = L(MaxToken) - L(InputPrompt) - L(Description_i) \quad (2)$$

Where $Description_i$ represents the brief description generated by LLM for the i^{th} candidate webshell sample, which is used to summarize the idea of candidate webshell generation and facilitate the subsequent voting process. This approach maximizes the length of the generated candidate webshell sample at the expense of consuming more token resources.

3.3.3 State Evaluator $V(M, O)$

Corresponding to Thought Generator, State Evaluator is also designed to have 2 different voting methods for large and small webshells. For small webshells, Hybrid Prompt uses LLM to vote on multiple intermediate webshell samples (states) and filter out the optimal ones. The reason for voting on multiple samples instead of voting on solutions is two-fold: 1) Since the Thought Generator operates in a few-shot CoT mindset, webshell samples help LLM evaluate and assess the differences between generated examples more intuitively to make optimal judgments; 2) Voting directly on the samples can preserve all the original information of the candidate webshells.

In this case, $L(Generator(Input + Output)) \approx L(Evaluator(Input + Output)) < L(MaxToken)$. Because both contain F_e, the webshell contents of p candidates, and additional prompt information.

For large webshells, it is not feasible to directly input the webshell contents of p candidates into LLM because $p \times (L(Candidate_i)) + L(F_e chain) + L(Additional_{Prompt}) > L(MaxToken)$. Therefore, we use $Description_i$ instead of $Candidate_i$ as the input component of the voting procedure. This kind of information compression idea will inevitably lose the original code information. App. G presents a specific example comparing 2 voting ideas.

Regardless of the voting idea, for $V(M, O)$, where $O = \{o_1, o_2, ..., o_p\}$, $V(M, o_i) = 1$ is considered a good state, when $o_i \sim M^{vote}(o_i|O)$. For Hybrid Prompt, the evaluation of a good state is to synthesize both the confusion level of the intermediate results generated by LLM for a module and the distance between them and the F_es. By allowing LLM to pursue local optimal solutions at each step of sample generation, this "greedy" idea makes it easier for the LLM to approximate the global optimal solution for the heuristic problem of escape sample generation.

3.3.4 Search Algorithm

For the Hybrid Prompt method, the depth of the tree N corresponds to the total number of modules. The DFS strategy leads to an excessive state space of LLM during the backtracking and pruning stages, which reduces the efficiency of the algorithm operation. Therefore, we consider using the BFS search algorithm. The pseudocode of the corresponding Hybrid Prompt-BFS algorithm is shown in Algorithm 2.

Algorithm 2 Hybrid Prompt-BFS Algorithm

Require: Input x, Thought Generator $G(M, o)$, State Evaluator $V(M, O)$, Tree Depth N, Candidate num p, Step Output $O_i (O \leq i \leq N)$

$O_0 = x$
for n = 1 to N **do**
  $O_n^{'} = \{[o, z] | o \in O_{n-1}, z_n \in G(M, o)\}$
  $V_n = V(M, O_n^{'})$
  $O_n = sort(V_n, p)$
end for
Return O_n

Taking into account performance and efficiency considerations, for the escape sample generation task, we set the number of candidates p to 1. The final output of the webshell escape sample is the candidate that wins in the vote process at the N^{th} layer.

3.3.5 Contextual Memory Range

Since LLM has a limited range of contextual memory, we cannot let LLM memorize the entire Hybrid Prompt context but should set its local memory range. For this reason, our approach is to set the Contextual Memory Range for the Hybrid Prompt, as shown in Figure 3. See more explanations in App. H.

3.3.6 Additional Explanation

For the webshell escape sample generation task, an important guiding principle is to ensure the validity of generated samples. This means that the escaped samples should not lose the attack behavior and malicious features of the original samples and can be executed correctly without any syntax or lexical errors. To achieve this, Hybrid Prompt introduces Safeguard Prompt to constrain sample generation and improve SR. In addition, common techniques in prompt engineering, such as "‘ delimiter, are also applied in the Hybrid Prompt algorithm to normalize the output of LLMs.

The order of modules also has a significant impact on the Hybrid Prompt algorithm. Therefore, when running the Hybrid Prompt algorithm, it is important to consider the relative position between specific modules and establish corresponding rules

to avoid such situations from occurring. See details in App. I.

4 Experiments

4.1 Setup

In the experimental section, our main objective is to answer the following questions:

RQ1: Can LLM effectively generate escape samples, and what is the ER of these samples under different detection engines?

RQ2: Are the individual parts of the Hybrid Prompt algorithm effectively designed?

RQ3: Does the number of candidates p affect the performance of the Hybrid Prompt algorithm?

Experimental Environment. The specific experimental environment is shown in Table 1. See details in App. J.

CPU	Intel Xeon(R) Gold 6326 CPU @ 2.9GHz
RAM	64GB
GPU	NVIDIA TESLA A100-SXM4-80G $\times 2$
Language	Python 3.10+
AI Framework	PyTorch 1.8.1+
Virtual Attack Environment	DVWA + AntSword

Table 1: Experimental Environment

Evaluation Metrics. To better compare the quality of samples generated by different LLM models using the Hybrid Prompt algorithm, we choose two evaluation metrics: ER and SR, which are calculated as follows:

$$ER = 1 - DR = 1 - N_{Detected\_samples}/N_{Total\_samples} \quad (3)$$

$$SR = N_{Malicious\_samples}/N_{Total\_samples} \quad (4)$$

Where Detection Rate (DR) represents the detection accuracy of the detection engine, $N_{Total\_samples}$ is the total number of samples generated by LLM under the Hybrid Prompt algorithm, $N_{Detected\_samples}$ is the number of webshells successfully identified by the detection engine, and $N_{Malicious\_samples}$ is the number of samples generated by LLM under the Hybrid Prompt algorithm that still retain malicious functionality.

Models & Detection Engines. We test the ER and SR of samples generated by Hybrid Prompt under three detection engines: Web Shell Detector, WEBDIR+, and VIRUSTOTAL respectively. By calling the VIRUSTOTAL scanning API, we test more than 58 different detection engines (i.e. AVG, ClamAV, AVAST, etc.). In addition, we cross-check the performance of several LLM models, including GPT-3.5, GPT-4, and Code-llama-34B, which demonstrate excellent performance in code generation and semantic understanding tasks.

Comparative Methods. Due to the lack of relevant research, we also include a comparison with the dataset from CWSOGG (Pang et al., 2023), an obfuscated webshell dataset generated using the genetic algorithm.

4.2 Comparative Experiment

To answer **RQ1**, the comparative results are shown in Table 2.

In Table 2, the GPT-4 + Hybrid Prompt algorithm has the best comprehensive performance, leading to both ER and SR. This is due to the fact that GPT-4 is more capable of following complex instructions carefully, while Hybrid Prompt contains multiple detailed instructions with normalized constraints. GPT-3.5, on the other hand, could partially follow complex instructions, resulting in a higher probability of generating escape samples that prioritize either ER or SR, making it difficult to balance both. It is encouraging to note that the comprehensive performance of the open-source LLM Code-llama-34B, is very close to that of the GPT-3.5 model, confirming the performance potential of the open-source models. Meanwhile, the ER of webshell samples generated by the 3 LLM models + Hybrid Prompt algorithm have far exceeded those of the Original Template Dataset and the CWSOGG Dataset, which fully demonstrate the performance superiority and dominance of the LLM models over rule-based artificial escape strategies and the traditional intelligent algorithms (i.e., genetic algorithm). As for the detection engines, VIRUSTOTAL, due to its integration of many different detection engines, has a higher overall DR compared to Web Shell Detector and WEBDIR+. However, even VIRUSTOTAL struggles with the creativity of LLMs and the uncertainty of the generated escape samples, which illustrates the limitations and drawbacks of these type of specific rule-based detection engines. See App. K for visualization results.

4.3 Ablation Analysis

To further validate the effectiveness of the Hybrid Prompt algorithm and address **RQ2**, we test the performance of sample sets generated by removing different components of Hybrid Prompt under ER and SR evaluation metrics in the GPT-3.5 model. Specifically, we refer to the complete Hybrid Prompt algorithm as Strategy 1, removing the

Anti-Virus Engine Model	*Web Shell Detector*	*WEBDIR+*	*VIRUSTOTAL*	SR
	ER			
GPT-3.5 Turbo + Hybrid Prompt	0.9342	0.8874	0.7465	0.4093
GPT-4 + Hybrid Prompt	0.9727	0.9287	0.8861	0.5498
Code-llama-34B + Hybrid Prompt	0.9015	0.8549	0.6358	0.3021
Original Template Dataset	0.3415	0.2054	0.1232	1
CWSOGG Dataset	0.4052	0.3151	0.2327	1

Table 2: Comparative Experiment Results

Safeguard Prompt as Strategy 2, removing F_e chain as Strategy 3, removing the voting strategy and generating only 1 sample per module as Strategy 4, and letting the LLM directly generate webshell as Strategy 5. The experimental results are shown in Table 3.

Table 3 illustrates that Strategy 5 has poor performance and a high probability of hallucination due to the absence of any additional prompt. Both Strategy 3 and Strategy 4 produce different degrees of performance degradation. For Strategy 3, LLM loses reference examples, leading to a higher probability of generating corrupted samples. Strategy 3 also indirectly reflects that the current LLM's code reasoning ability still relies on F_e chains to achieve better task performance. For Strategy 4, LLM is unable to explore multiple reasoning paths, so the generation space and diversity of samples are limited, which leads to a lower ER. Strategy 2 has the least impact on the quality of generated escape samples. Although the probability of generating corrupted samples increases and the SR decreases due to the loss of Safeguard Prompt's normalization measures, the impact on the ER is not significant. However, for Strategy 2 - Strategy 5, all produce varying degrees of performance degradation compared to the complete Hybrid Prompt algorithm, fully demonstrating the effectiveness of various components of the Hybrid Prompt algorithm. See App. K for visualization results.

4.4 Sensitivity Analysis

We investigate the impact of the candidate number, p, on the SR and ER evaluation metrics of generated samples in the GPT-3.5 model. The experimental results of **RQ3** are shown in Table 4.

From Table 4, it can be observed that a larger number of candidates can increase the search space of LLM, which in turn enriches the diversity of generated samples, enables better selection of the optimal solution, and improves the sample ER and SR. However, the increase of p will also result in a higher token consumption and, in the case of small webshells, further reduces the $L(Avg\_Candidate_i)$ for each sample. Figure 4 is able to visualize the "marginal effect" that occurs as p increases (The yellow and purple folds in Figure 4 almost overlap). When p exceeds 3, the performance improvement of ER and SR metrics is not obvious, which can be attributed to the fact that the search space of LLM's self-inference is approaching the local upper limit. However, it is noteworthy that the consumption of tokens exhibits an almost linear relationship with the increase in p, despite the limited performance gains in ER and SR metrics. Therefore, the pros and cons between evaluation metrics and resource consumption should be weighed in practical applications.

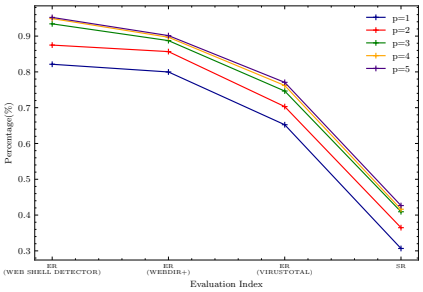

Figure 4: Visualization of the "marginal effect" with increasing p

5 Related Work

5.1 Prompt Engineering Algorithm.

As one of the most classic prompt algorithms, CoT (Wei et al., 2022) aims to assist LLMs in achieving complex reasoning abilities through intermediate inference steps. Zero-shot CoT (Kojima et al., 2022), as a follow-up to CoT, enables LLM to perform self-reasoning through twice generation, involving 2 separate prompting processes. SC (Wang et al., 2023) serves as another complement to the CoT algorithm by sampling a diverse set of reasoning paths and marginalizing out reasoning paths to aggregate final answers. Least to Most Prompting (LtM) (Zhou et al., 2023), also an advancement of the CoT algorithm, decomposes a problem into a set of subproblems built upon each other and in-

Anti-Virus Engine Strategy	*Web Shell Detector*	*WEBDIR+*	*VIRUSTOTAL*	SR
	ER			
Strategy 1	0.9342	0.8874	0.7465	0.4093
Strategy 2	0.9221	0.8653	0.7114	0.3398
Strategy 3	0.7315	0.6819	0.5042	0.2310
Strategy 4	0.8213	0.7998	0.6524	0.3067
Strategy 5	0.5021	0.4267	0.3120	0.1513

Table 3: The Comparative Results of Ablation Analysis

Anti-Virus Engine Candidate num p	*Web Shell Detector*	*WEBDIR+*	*VIRUSTOTAL*	SR
	ER			
1	0.8213	0.7998	0.6524	0.3067
2	0.8749	0.8567	0.7031	0.3648
3	0.9342	0.8874	0.7465	0.4093
4	0.9489	0.8968	0.7621	0.4163
5	0.9522	0.9014	0.7708	0.4266

Table 4: The Comparative Results Of Sensitivity Analysis

puts the solutions of the previous sub-problem into the prompt of the next sub-problem to gradually solve each sub-problem. Generated Knowledge Approach (GKA) (Liu et al., 2022) enables LLM to generate potentially useful information related to a given question before generating the response through 2 intermediate steps: knowledge generation and knowledge integration. Diverse Verifier on Reasoning Steps (DiVeRSe) (Li et al., 2023), on the other hand, improves the reliability of LLM answers by generating multiple reasoning paths.

5.2 The Application Of LLM In Code Related Tasks.

Zhang et al. (Zhang et al., 2023a) utilized ChatGPT to generate vulnerability exploitation code. Liu et al. (Liu et al., 2023b) applied GPT to the task of vulnerability description mapping and evaluation tasks. They provided certain prompts to ChatGPT and extracted the required information from its responses using regular expressions. Zhang et al. (Zhang et al., 2023b) proposed STEAM, a framework for bug fixing using LLM to simulate programmers' behaviors. Kang et al. introduced the LIBRO (Kang et al., 2023) model for exploring bug reproduction tasks. The aforementioned researches demonstrate that with appropriate algorithmic design, LLM is capable of handling various specific tasks in the field of code analysis.

5.3 Researches On Webshell Detection Techniques.

We categorize the research in the field of webshell detection into 3 stages: Start Stage, Initial Development Stage, and In-depth Development Stage. In the Start Stage, research methods are simple and have numerous flaws and deficiencies, such as limited private datasets, unreasonable feature extraction methods, oversimplified classifier structure design (Tian et al., 2017; Zhang et al., 2018), etc. In the Initial Development Stage, relevant studies explore and make progress in various aspects of the detection process. However, theoretical innovations remain relatively scarce (Wu et al., 2019; Lu et al., 2020; Zhang et al., 2020; Le et al., 2023; Zhou et al., 2021), etc. In the In-depth Development Stage, simple individual classifiers or machine learning algorithms become less common, and related research has penetrated into the theoretical process level of modeling methods (An et al., 2022; Cheng et al., 2022). However, from an overall point of view, research related to webshell detection techniques is still in its early stages, largely due to the slow progress of the attacker's research, and the lack of advanced webshell escape sample generation algorithms in the field.

6 Conclusion

In this paper, we propose Hybrid Prompt, a webshell escape sample generation prompt algorithm that combines various prompt strategies such as ToT, CoT, etc. Hybrid Prompt combines structured webshell module and F_e chain, utilizes auxiliary methods to inspire LLMs to perform self-assessment and optimization, and demonstrates excellent performance on LLMs with strong code reasoning capabilities (GPT-3.5, GPT-4, Code-llama-34B), enabling the generation of high-quality webshell escape samples. Hybrid Prompt algorithm also exhibits strong scalability and generalization capability, allowing for the addition of more modules and corresponding F_e chains to update escape strategies and expand to more webshell languages.

7 Limitations

1. The Hybrid Prompt algorithm currently supports a limited number of webshell languages, and there is a need to expand it to support more webshell languages in the future.

2. Hybrid Prompt algorithm does not fine-tune LLMs. Fine-tuning can further reduce the probability of LLM hallucination and improve the quality of generated escape samples.

3. For the voting strategy in the case of large webshells, the description-based strategy used in the Hybrid Prompt algorithm results in the loss of original information from candidate code, which in turn affects the vote effect of LLM. While information compression strategies are acceptable for NLP tasks such as contextual dialogs, there is room for further improvement for tasks such as code generation, which require precise raw sample information.

Therefore, our further work includes combining LLM fine-tuning techniques with the Hybrid Prompt algorithm to further enhance the code generation capability of LLM and designing more advanced information compression algorithms to improve the quality of sample generation.

8 Ethics Statement

All experiments in this paper (Section 4) are conducted under the built Virtual Attack Environment, thus posing no harm to the real internet environment. Additionally, the algorithms and data included in this work are intended to contribute to the development and transformation of webshell detection techniques, solely for academic research reference, and are strictly prohibited for any real-world cyber-attack activities. Beyond that, we believe that this research does not produce any other potential harm or bias.

References

Tongjian An, Xuefei Shui, and Hongkui Gao. 2022. Deep learning based webshell detection coping with long text and lexical ambiguity. In *Information and Communications Security - 24th International Conference, ICICS 2022, Canterbury, UK, September 5-8, 2022, Proceedings*, volume 13407 of *Lecture Notes in Computer Science*, pages 438–457. Springer.

Raphael Labaca Castro, Battista Biggio, and Gabi Dreo Rodosek. 2019. Poster: Attacking malware classifiers by crafting gradient-attacks that preserve functionality. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2565–2567. ACM.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2023. A survey on evaluation of large language models. *CoRR*, abs/2307.03109.

Baijun Cheng, Yanhui Guo, Yan Ren, Gang Yang, and Guosheng Xu. 2022. Msdetector: A static PHP webshell detection system based on deep-learning. In *Theoretical Aspects of Software Engineering - 16th International Symposium, TASE 2022, Cluj-Napoca, Romania, July 8-10, 2022, Proceedings*, volume 13299 of *Lecture Notes in Computer Science*, pages 155–172. Springer.

Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Trans. Inf. Forensics Secur.*, 16:3469–3478.

Gelei Deng, Yi Liu, Victor Mayoral Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. 2023. Pentestgpt: An llm-empowered automatic penetration testing tool. *CoRR*, abs/2308.06782.

Rahul Dey and Fathi M. Salem. 2017. Gate-variants of gated recurrent unit (GRU) neural networks. In *IEEE 60th International Midwest Symposium on Circuits and Systems, MWSCAS 2017, Boston, MA, USA, August 6-9, 2017*, pages 1597–1600. IEEE.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: general language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 320–335. Association for Computational Linguistics.

Abdelhakim Hannousse and Salima Yahiouche. 2021. Handling webshell attacks: A systematic mapping and survey. *Comput. Secur.*, 108:102366.

Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring llm-based general bug reproduction. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 2312–2323. IEEE.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *26th European Signal Processing Conference, EUSIPCO 2018, Roma, Italy, September 3-7, 2018*, pages 533–537. IEEE.

Ha Viet Le, Tu N. Nguyen, Hoa Ngoc Nguyen, and Linh Le. 2023. An efficient hybrid webshell detection method for webserver of marine transportation systems. *IEEE Trans. Intell. Transp. Syst.*, 24(2):2630–2642.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 5315–5333. Association for Computational Linguistics.

Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. 2023a. Improving chatgpt prompt for code generation. *CoRR*, abs/2305.08360.

Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. 2022. Generated knowledge prompting for commonsense reasoning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3154–3169. Association for Computational Linguistics.

Xin Liu, Yuan Tan, Zhenghang Xiao, Jianwei Zhuge, and Rui Zhou. 2023b. Not the end of story: An evaluation of chatgpt-driven vulnerability description mappings. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 3724–3731. Association for Computational Linguistics.

Jinping Lu, Zhi Tang, Jian Mao, Zhiling Gu, and Jiemin Zhang. 2020. Mixed-models method based on machine learning in detecting webshell attack. In *CIPAE 2020: 2020 International Conference on Computers, Information Processing and Advanced Education, Ottawa, ON, Canada, October 16-18, 2020*, pages 251–259. ACM.

Bo Pang, Gang Liang, Jin Yang, Yijing Chen, Xinyi Wang, and Wenbo He. 2023. CWSOGG: catching web shell obfuscation based on genetic algorithm and generative adversarial network. *Comput. J.*, 66(5):1295–1309.

Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. 2023. Examining zero-shot vulnerability repair with large language models. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 2339–2356. IEEE.

Peng Peng, Limin Yang, Linhai Song, and Gang Wang. 2019. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019*, pages 478–485. ACM.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2463–2473. Association for Computational Linguistics.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 4222–4235. Association for Computational Linguistics.

Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2022. Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware. In *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*, pages 990–1003. ACM.

Oleksii Starov, Johannes Dahse, Syed Sharique Ahmad, Thorsten Holz, and Nick Nikiforakis. 2016. No honor among thieves: A large-scale analysis of malicious web shells. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 1021–1032. ACM.

Ralf C. Staudemeyer and Eric Rothstein Morris. 2019. Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, abs/1909.09586.

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, and Yang Liu. 2023. When GPT meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in gptscan. *CoRR*, abs/2308.03314.

Yifan Tian, Jiabao Wang, Zhenji Zhou, and Shengli Zhou. 2017. Cnn-webshell: Malicious web shell detection with convolutional neural network. In *Proceedings of the VI International Conference on Network, Communication and Computing, ICNCC 2017, Kunming, China, December 8-10, 2017*, pages 75–79. ACM.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yuxiang Wei, Chunqiu Steven Xia, and Lingming Zhang. 2023. Copiloting the copilots: Fusing large language models with completion engines for automated program repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 172–184. ACM.

Yixin Wu, Yuqiang Sun, Cheng Huang, Peng Jia, and Luping Liu. 2019. Session-based webshell detection using machine learning in web logs. *Secur. Commun. Networks*, 2019:3093809:1–3093809:11.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *CoRR*, abs/2305.10601.

Han Zhang, Ming Liu, Zihan Yue, Zhi Xue, Yong Shi, and Xiangjian He. 2020. A PHP and JSP web shell detection system with text processing based on machine learning. In *19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020, Guangzhou, China, December 29, 2020 - January 1, 2021*, pages 1584–1591. IEEE.

Ying Zhang, Wenjia Song, Zhengjie Ji, Danfeng Yao, and Na Meng. 2023a. How well does LLM generate security tests? *CoRR*, abs/2310.00710.

Yuwei Zhang, Zhi Jin, Ying Xing, and Ge Li. 2023b. STEAM: simulating the interactive behavior of programmers for automatic bug fixing. *CoRR*, abs/2308.14460.

Zijian Zhang, Meng Li, Liehuang Zhu, and Xinyi Li. 2018. Smartdetect: A smart detection scheme for malicious web shell codes via ensemble learning. In *Smart Computing and Communication - Third International Conference, SmartCom 2018, Tokyo, Japan, December 10-12, 2018, Proceedings*, volume 11344 of *Lecture Notes in Computer Science*, pages 196–205. Springer.

Jianyu Zhao, Yuyang Rong, Yiwen Guo, Yifeng He, and Hao Chen. 2023a. Understanding programs by exploiting (fuzzing) test cases. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10667–10679. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023b. A survey of large language models. *CoRR*, abs/2303.18223.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Ziheng Zhou, Lin Li, and Xu Zhao. 2021. Webshell detection technology based on deep learning. In *7th IEEE International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security, BigDataSecurity/HPSC/IDS 2021, New York City, NY, USA, May 15-17, 2021*, pages 52–56. IEEE.

Weiming Zhuang, Chen Chen, and Lingjuan Lyu. 2023. When foundation model meets federated learning: Motivations, challenges, and future directions. *CoRR*, abs/2306.15546.

A VIRUSTOTAL achieves high-precision detecton of open-access webshell repositories

For a limited number of publicly available webshell repositories on the internet, detection engines can also achieve high-precision detection, and the superiority of artificial intelligence-based methods is not fully demonstrated. We apply the VIRUSTOTAL detection engine to various open-access webshell repositories on GitHub, achieving high-precision detection of different webshells. Figure 5

gives a specific example of VIRUSTOTAL detecting the "tennc/webshell" [3] repository.

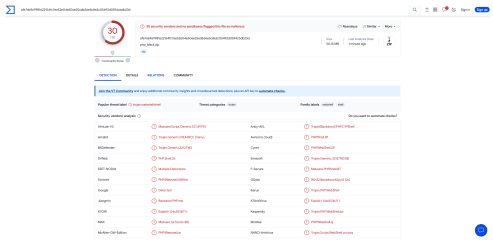

Figure 5: VIRUSTOTAL achieves high precision detection

B LLM generates new webshells

By providing a simple prompt and giving a rough idea of webshell escape techniques, LLM can generate a 0-day webshell (Figure 6). However, in such a situation, the lack of strict prompt constraints and complete thought flows can lead LLM to generate a potentially corrupted webshell.

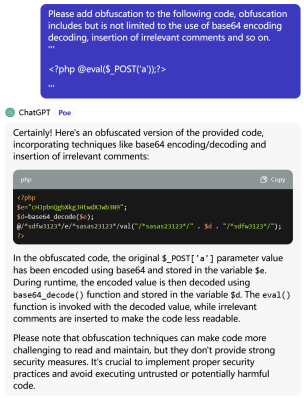

Figure 6: Generating webshells using simple prompts on LLM

C Toy tests on some lightweight LLMs

LLMs like Chatglm and Chatglm2, perform poorly on the webshell escape sample generation task due to their weak code reasoning abilities (Figure 7). Even when adjusting key parameters such as $Temperature$, Top_p, Top_k, etc., or even fine-tuning such models, the results still yield little effect. The fundamental reasons are two-fold. Firstly, Chatglm and other LLM models focusing on interactive dialogs have weak reasoning ability, while the webshell escape sample generation task requires strong inference ability. (The model should effectively understand each specific escape strategy in the prompts and modify the given examples for bypassing without destroying the original functionality and syntactic structure of the webshell.) Secondly, prompt engineering itself tends to have more significant effects on LLMs with more than 30B parameters.

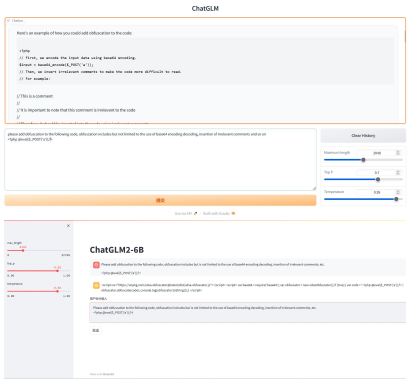

Figure 7: Chatglm and Chatglm2 models perform poorly on the task of webshell generation

D Examples in the triple data filtering process

Figure 8 gives a specific example of the AST structure generated for a small webshell. Each information node in the tree contains "name" and "kind" attributes. Figure 9 illustrates an example of php opcodes.

E Hierarchial module structure

This hierarchical structure of modules in Figure 10 constitutes a forest structure, in which each primary module is the root node of the tree in the forest. This modular design concept has strong scalability, allowing for the real-time addition of modules to increase the number of escape methods for the Hybrid Prompt algorithm.

F F_e chain structure

Figure 11 provides a specific example of the F_e chain in the "Array methods" module. This chain structure helps LLM to rapidly grasp the core escape ideas within the corresponding module.

G Example of 2 different voting strategies

Figure 12 presents a concrete example of 2 different voting strategies. Left: Small Webshell's voting strategy, where all raw webshell information is contained in a single contextual dialog; Right: Large

[3] https://github.com/tennc/webshell

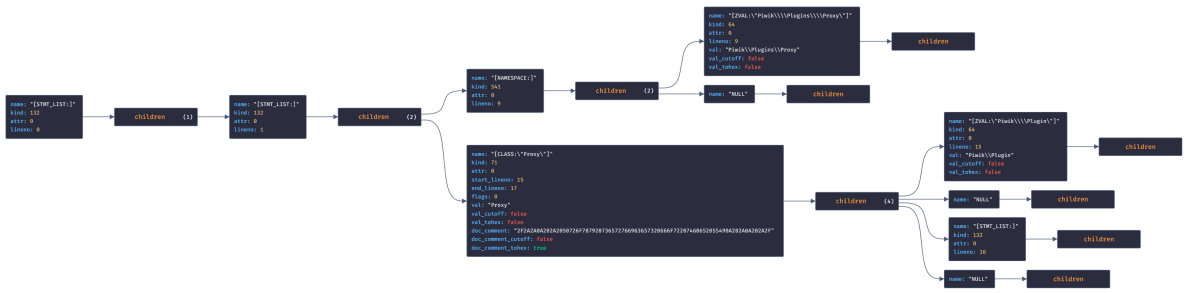

Figure 8: An example of webshell AST structure

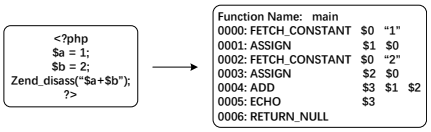

Figure 9: A typical example of generating opcodes through VLD disassembler

Webshell's voting strategy, where information is compressed for every candidate generated by LLM.

H Explanation of Contextual Memory Range

For Hybrid Prompt itself, it is impossible to compress the history information like many NLP tasks (e.g. contextual conversations) because it would result in a significant loss of raw webshell information. Hence, Contextual Memory Range refers to the scope of each iteration in the Hybrid Prompt algorithm. At this stage, the only contextual information required for the next iteration round is the candidate output selected by the winning vote strategy in the previous iteration. Therefore, defining the Contextual Memory Range ensures the continuity of information memory throughout the complete Hybrid Prompt algorithm. Correspondingly, O_n', V_n within the body of the "for" loop in the Algorithm 2 are the local contextual contents that LLM needs to memorize.

I Effect of module order on the Hybrid Prompt algorithm

In Figure 13, if the "String XOR Encryption" module is placed in front of the "Symbol Interference" module, the encrypted webshell sample is no longer "text-readable", resulting in a high probability of hallucination when LLM executes to the "Symbol Interference" module, and triggering a series of subsequent generation errors. Therefore, during the implementation of Hybrid Prompt algorithm, we strictly constrain the relative positions between different modules.

J Virtual attack environment

Specifically, we use a virtual environment simulating a vulnerable server in DVWA and apply AntSword virtual environment for attack testing. In Figure 14, the attacker exploits vulnerabilities in the DVWA server to perform a File Upload operation and implant a webshell file. Subsequently, the attacker utilizes the remote connection feature of the webshell file in AntSword to gain operational privileges on the DVWA server and execute malicious behaviors.

K Supplementary materials for the experimental results

Additional Explanations. By default, we set the number of candidates p to 3. Due to the frequent updating and maintenance of detection engines, the actual test results may differ slightly from the results presented in this paper. However, the experimental results can still effectively reflect the performance differences and data trends among different methods.

Figure 15 and Figure 16 visualize the performance differences as reflected in Table 2 and Table 3 respectively.

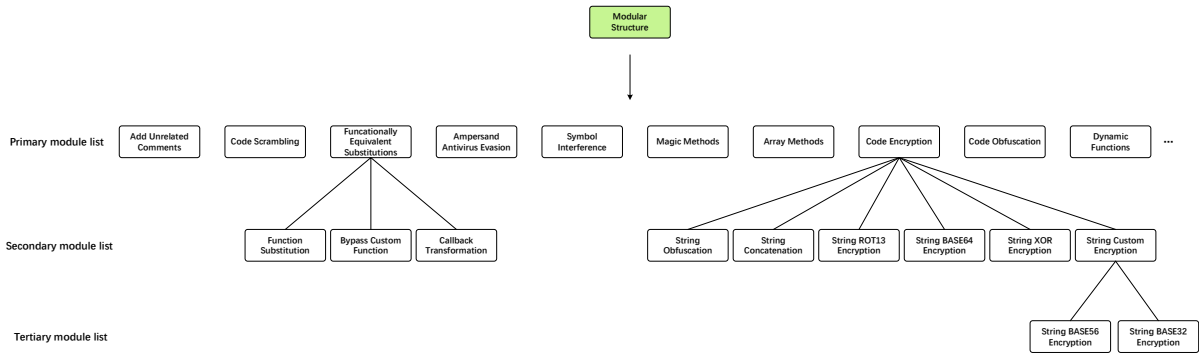

Figure 10: Hierarchial module structure in Hybrid Prompt

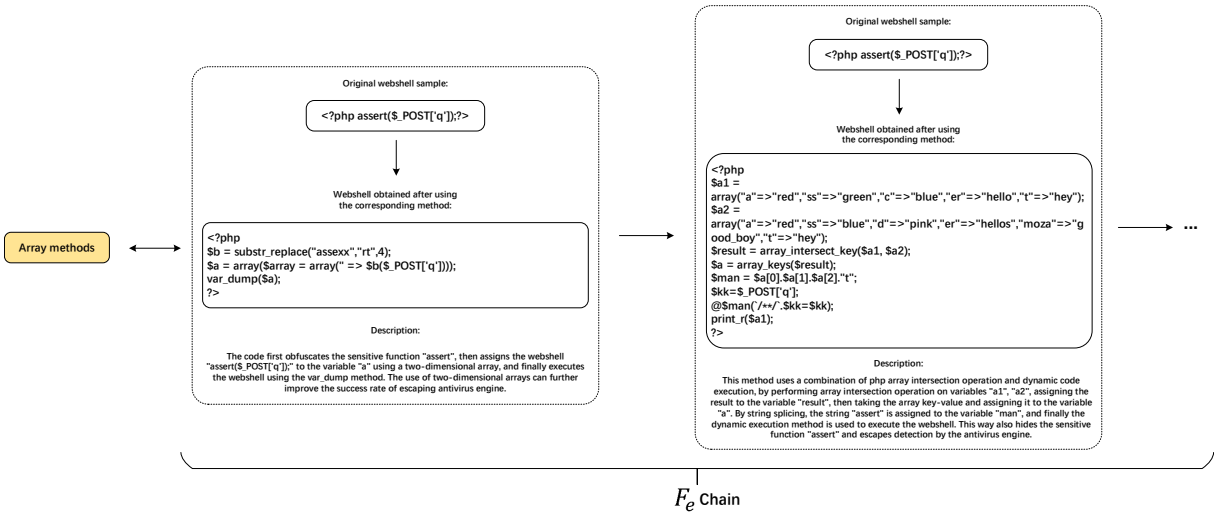

Figure 11: The structure of F_e chain for each module

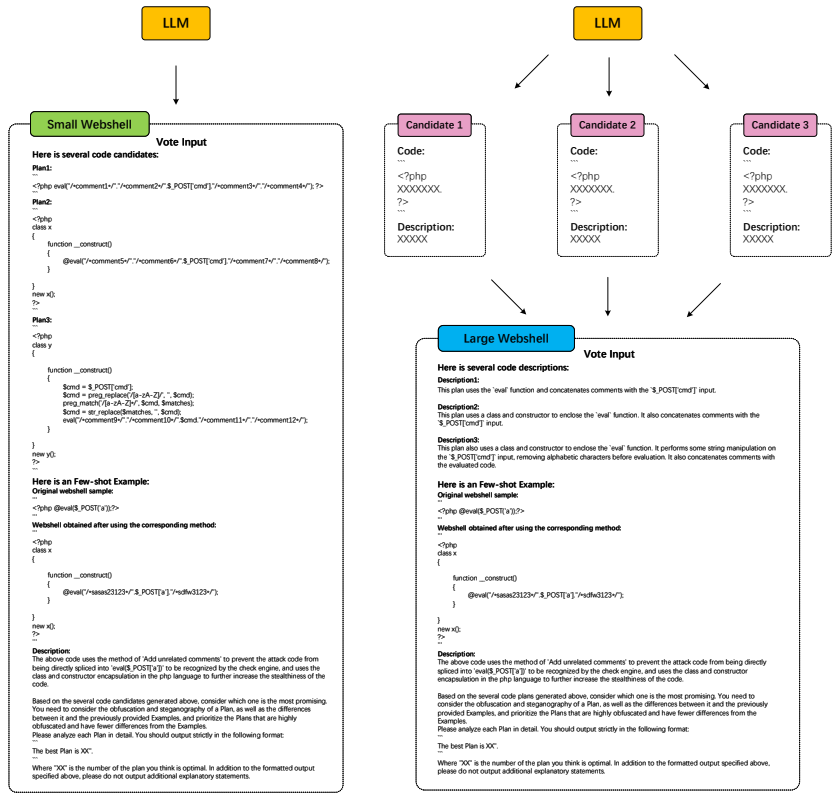

Figure 12: Comparison of 2 different vote ideas

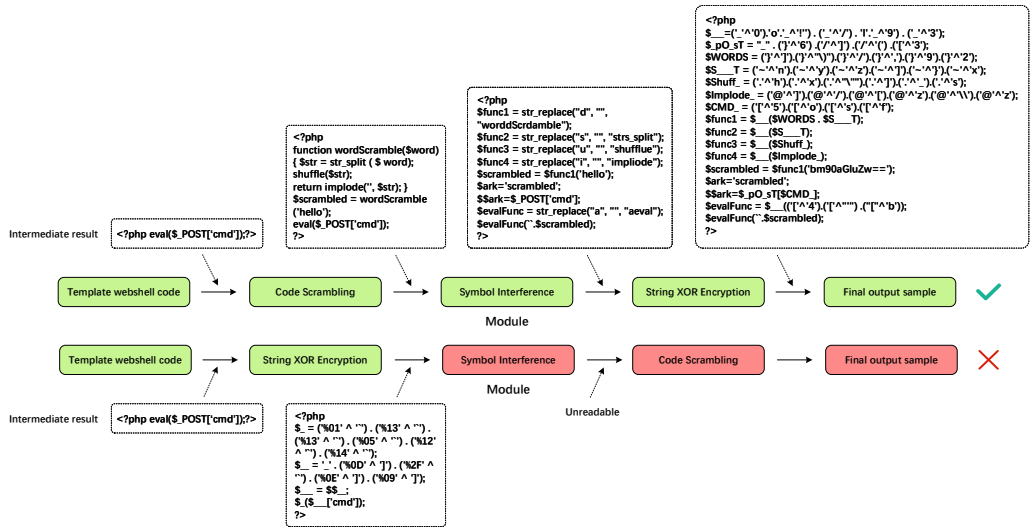

Figure 13: Effect of module order on the Hybrid Prompt algorithm (Incorrect module order can result in abnormal output from the LLM)

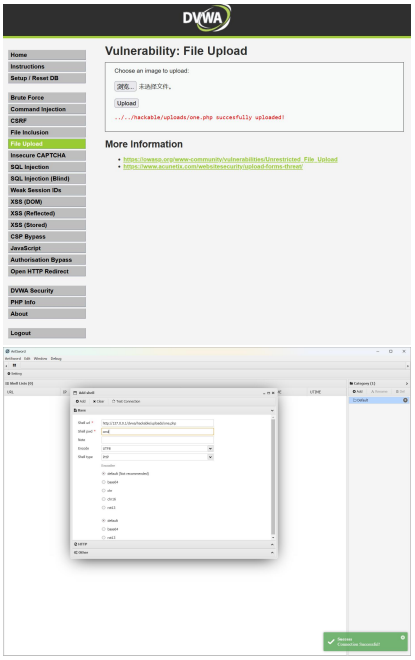

Figure 14: Virtual attack environment. Above: DVWA server; Below: AntSword attack interface.

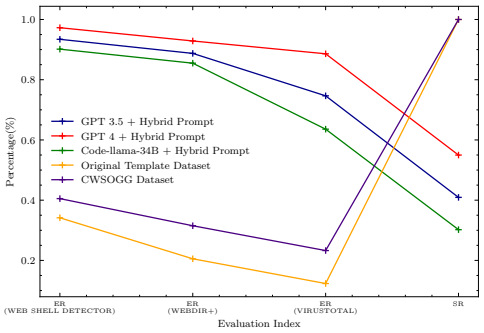

Figure 15: Performance comparison of different LLM models on Hybrid Prompt algorithm

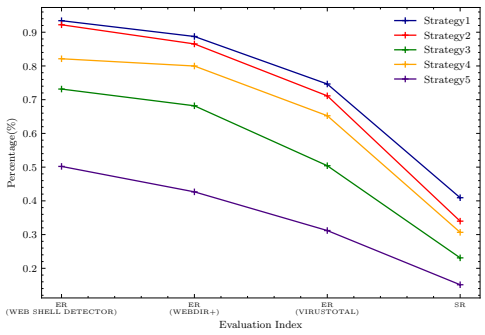

Figure 16: Visualization of ablation analysis results for Hybrid Prompt algorithm