LPDetective: Dusting the LLM Chats for Prompt Template Abusers

Yang Luo^{1,2,3} , **Qingni Shen**^{1,2,3} and **Zhonghai Wu**^{1,2,3*}
^{1}National Engineering Research Center for Software Engineering, Peking University, Beijing, China
^{2}School of Software and Microelectronics, Peking University, Beijing, China
^{3}PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University, Beijing, China
{luoyang, qingnishen, wuzh}@pku.edu.cn

Abstract

The abuse of LLM Chatbot interfaces by web robots leads to a significant waste of GPU and server resources, posing a serious security challenge. To address this issue, we propose LPDetective, an unsupervised method for detecting robot prompt templates. This method is based on the assumption that robot-generated text repeatedly uses the same or highly similar phrases and sentence structures across multiple sessions, differing from human natural conversations. We design a multi-stage workflow, including message grouping, text similarity measurement, hierarchical clustering analysis, and regular expression extraction, to automatically extract potential robot behavior patterns from chat logs. LPDetective does not require predefined templates or rely on training data, enabling it to adaptively discover new, unknown patterns. We conduct systematic experiments on three large-scale real-world datasets: Bing Copilot, Wildchat, and ChatLog. The results show that LPDetective can efficiently and accurately detect robot prompt templates in various scenarios, achieving a 7.5% improvement in F1 score compared to the state-of-the-art XLNet method and reducing detection latency by 178 times on the Bing Copilot dataset.

1 Introduction

In recent years, chatbot technology driven by large language models (LLMs) has made tremendous progress, such as ChatGPT [Ouyang *et al.*, 2022; Wikipedia, 2022], GPT-4 [OpenAI, 2023], etc., exhibiting near-human or even superhuman capabilities in natural conversation, question answering, and content creation [Frieder *et al.*, 2023; McGee, 2023]. These chatbots have been widely applied in customer service [Følstad and Skjuve, 2019], education [Wollny *et al.*, 2021], healthcare [Ayanouz *et al.*, 2020], and other fields, profoundly changing our work and lifestyles [Hussain *et al.*, 2019; Gupta *et al.*, 2020; Paliwal *et al.*, 2020].

However, the open interaction interfaces of LLM chatbots also introduce new security risks. In 2023, OWASP listed Model Denial of Service (MDoS) attacks as the 4th biggest threat faced by LLM systems [OWASP, 2023]. In MDoS attacks, malicious attackers can use automated tools to generate conversation requests in bulk, rapidly consuming the computing and storage resources of LLM systems, affecting the experience of normal users, and even leading to complete service unavailability. For example, in November 2023, OpenAI's ChatGPT service suffered a targeted distributed denial of service (DDoS) attack, causing the system to be down for an extended period [CNBC, 2023]. In August of the same year, security researchers successfully constructed malicious conversation requests by reverse engineering the API interface of the new Bing Chat, confirming its risk of MDoS abuse [Integration-Automation, 2023].

To counter LLM robot abuse attacks, academia and industry have proposed a series of detection and defense methods, mainly focusing on strategies such as adversarial training [Xu *et al.*, 2021], access control [Gondaliya *et al.*, 2020], and anomaly detection [Qian *et al.*, 2023]. However, these methods struggle to cope with increasingly complex abuse scenarios, especially the problem of Chatbot interfaces being massively abused by robots. Robots quickly call APIs to generate harmful content, seriously wasting computing power, a phenomenon that has become particularly prominent after Chatbots opened their APIs [Economist, 2023; Hurler, 2023]. To address this issue, this paper innovatively proposes utilizing unsupervised learning to automatically mine robot Prompt templates from historical chat logs and construct regular expressions for real-time detection. This method can adaptively handle template syntax changes, resist template obfuscation, and is computationally efficient, overcoming the limitations of existing work.

This paper proposes LPDetective, an unsupervised LLM robot detection method based on chat log mining. This method leverages the fact that robots often reuse similar prompt templates in bulk, leading to a certain level of similarity and repetition in the syntactic and semantic structures of conversations. We design a multi-stage chat log mining framework, including message grouping, text similarity calculation, hierarchical clustering, and regular expression extraction, to automatically discover potential robot prompt templates from massive human-machine conversation records and generate corresponding detection rules. Unlike existing methods, LPDetective does not require predefined template

[*]Corresponding author

libraries or manually labeled data and can adaptively mine new, unknown patterns.

We evaluate the performance of LPDetective on three real-world datasets: Bing Copilot, Wildchat [Zhao *et al.*, 2024], and ChatLog [Tu *et al.*, 2023]. The experimental results demonstrate that this method can effectively mine high-frequency robot prompt templates in multiple scenarios. For example, on 118,000 chat records in the Bing Copilot dataset, the regular expression rules generated by LPDetective achieve an average accuracy of 94.6%, recall of 93.4%, and F1 score of 94.0%, outperforming the best baseline method XLNet by 7.3%, 7.7%, and 7.5%, respectively, in the same hardware environment. Moreover, the detection latency is only 0.294 seconds, significantly lower than XLNet's 52.438 seconds.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces the overall process and key algorithms of LPDetective. Section 4 presents the experimental results. Section 5 discusses limitations and future work. Section 6 concludes the paper.

2 Related Work

In recent years, chatbot technology driven by large language models (LLMs) has made tremendous progress, but its rapid development has also brought numerous security risks [Van Dis *et al.*, 2023]. Malicious attackers may exploit Chatbots to generate harmful content, such as fake information [Shu *et al.*, 2020; Karanjai, 2022], malware [Mulgrew, 2023; VARINDIA, 2022], phishing emails [Baki *et al.*, 2017; Giaretta and Dragoni, 2020], etc. [Point, 2023]. Mulgrew et al. [Mulgrew, 2023] demonstrated how to use ChatGPT to write an undetectable data-stealing virus. Baki et al. [Baki *et al.*, 2017] found that utilizing natural language generation technology can automatically generate large-scale, personalized spoof emails, greatly enhancing the power of phishing attacks. Although Zellers et al. [Zellers *et al.*, 2019] proposed a defense method against neural fake news, and Stiff et al. [Stiff and Johansson, 2022] attempted to detect fake content generated by Chatbots, there are currently no mature and effective defense measures.

Furthermore, the security and privacy of Chatbots themselves are also threatened [Chung *et al.*, 2017]. Chung et al. [Chung *et al.*, 2017] discussed the security risks of voice assistants like Alexa. Some Chatbots have design flaws or software vulnerabilities that can be exploited by attackers [Lei *et al.*, 2017; Security, 2018]. Lei et al. [Lei *et al.*, 2017] systematically analyzed the vulnerabilities of home digital voice assistants using Alexa as an example. Moreover, users' private conversation data with Chatbots may be leaked [Agomuoh, 2022; Coles, 2023]. Recently, The Economist [Economist, 2023] reported that Samsung employees misused ChatGPT, leading to the disclosure of commercial secrets. The advice provided by Chatbots also lacks security reviews and may mislead users into dangerous behaviors [Prakken, 2020; Mijwil and Aljanabi, 2023]. Prakken [Prakken, 2020] pointed out that persuasive Chatbots may induce users to make inappropriate decisions when using crowd opinion graphs.

Existing work on Chatbot abuse mainly focuses on defense strategies such as adversarial training [Xu *et al.*, 2021], access control [Gondaliya *et al.*, 2020], and anomaly detection [Qian *et al.*, 2023]. Xu et al. [Xu *et al.*, 2021] proposed a method based on adversarial training to generate safer Chatbot responses. Qian et al. [Qian *et al.*, 2023] designed an anomaly detection method to identify out-of-bounds answers from Chatbots. However, these methods struggle to cope with increasingly complex abuse scenarios. In contrast, we focus on the problem of Chatbot interfaces being massively abused by robots. Robots quickly call APIs to generate harmful content, seriously wasting computing power [Economist, 2023; Hurler, 2023]. We innovatively propose utilizing unsupervised learning to automatically mine robot Prompt templates from historical chat logs and construct regular expressions for real-time detection. This method can adaptively handle template syntax changes, resist template obfuscation, and is computationally efficient, overcoming the limitations of existing work.

3 Methodology

3.1 Problem Definition

In this paper, we assume that the LLM chatbot platform to be protected provides public API interfaces similar to ChatGPT and Bing Copilot, which can be accessed without user login. The attacker has mastered the method of calling the platform's API and the parameter format, and their goal is to maximize the consumption of the LLM system's computing resources by sending a large number of malicious requests in batches, resulting in a decline in service quality or denial of service. We focus on attackers using reusable prompt templates to implement MDoS attacks, i.e., using tools to automatically generate a large number of homogeneous conversation requests. This approach can simplify the attack process and improve attack efficiency. Since the platform does not require user login, attackers can easily change IP addresses, User Agents, and other request parameters frequently to circumvent traditional rate limiting mechanisms. At the same time, the behavior of batch reuse of prompt templates also provides a unique entry point for our detection method. This paper fully utilizes this feature to automatically identify potential robot prompt templates by mining real human-machine conversation logs, thereby achieving efficient detection of MDoS attacks.

This paper proposes an innovative method based on LPDetective for automatically detecting abnormal dialogue behavior of chatbots. Our method is based on the assumption that the prompt text generated by robots repeatedly uses the same or highly similar phrases and sentence structures across multiple sessions, which is significantly different from the pattern of human natural language. As shown in Figure 1, we designed a multi-stage workflow, including message grouping, text similarity measurement, hierarchical clustering analysis, and regular expression extraction.

3.2 Message Grouping

We first group messages according to the user's fingerprint features. Formally, given a set of messages $\mathcal{M} = m_1, m_2, \ldots, m_n$ and a set of users $\mathcal{U} = u_1, u_2, \ldots, u_k$, we

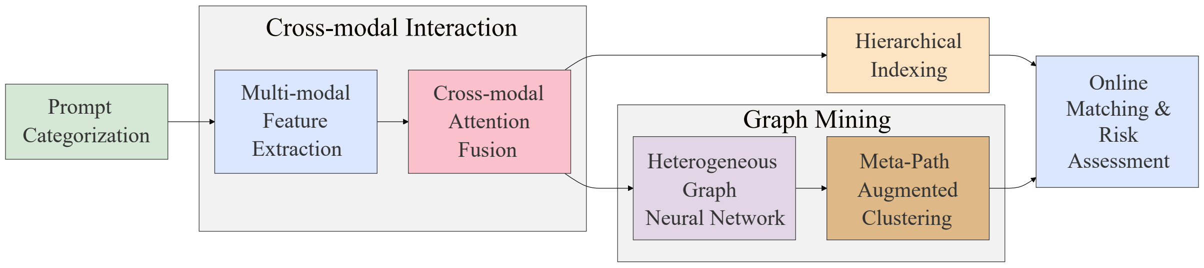
Prompt Categorization
Cross-modal Interaction
Multi-modal Feature Extraction
Cross-modal Attention Fusion
Hierarchical Indexing
Graph Mining
Heterogeneous Graph Neural Network
Meta-Path Augmented Clustering
Online Matching & Risk Assessment

Figure 1: Workflow of LPDetective.

define the fingerprint feature extraction function $\phi : \mathcal{U} \to 0, 1^d$, which maps each user to a d-dimensional binary vector. We divide the messages into different groups $\mathcal{G} = \{G_1, G_2, \ldots, G_l\}$ according to the user's fingerprint feature vector, where G_i contains all messages sent by user u_i.

To encode the fingerprint feature vector into a compact representation, we use the SimHash algorithm [Sadowski and Levin, 2007] for hashing. The SimHash algorithm maps high-dimensional feature vectors to low-dimensional binary codes, and the distance between codes is positively correlated with the distance between the original feature vectors.

Given a fingerprint feature vector $\mathbf{x} \in 0, 1^d$, the i-th bit of its SimHash code $\mathbf{c} \in 0, 1^b$ is calculated as follows:

$$c_i = \begin{cases} 1, & \text{if } \sum_{j=1}^{d} w_{ij} x_j \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where b is the number of bits in the SimHash code, and $w_{ij} \sim \mathcal{N}(0, 1)$ is an element of the random projection matrix.

3.3 Text Similarity Measurement

After obtaining the message groups $\mathcal{G} = \{G_1, G_2, \ldots, G_l\}$, we calculate the normalized Levenshtein distance [Levenshtein, 1966] for each pair of messages within each group to measure their text similarity. The normalized Levenshtein distance maps two strings to their edit distance under unit length:

$$d(s_1, s_2) = \frac{\text{lev}(s_1, s_2)}{\max(|s_1|, |s_2|)} \tag{2}$$

where $\text{lev}(\cdot, \cdot)$ is the Levenshtein distance, representing the minimum number of edit operations required to convert one string to another.

Based on the normalized Levenshtein distance, we can define the similarity matrix $S^{(i)} \in [0, 1]^{n_i \times n_i}$ for messages within group i, where $S^{(i)}_{jk} = 1 - d(m^{(i)}_j, m^{(i)}_k)$ represents the text similarity between messages $m^{(i)}_j$ and $m^{(i)}_k$ within group i.

3.4 Hierarchical Clustering

Based on the similarity matrix $\{S^{(1)}, S^{(2)}, \ldots, S^{(l)}\}$ for messages within each group, we use hierarchical clustering algorithms to further group messages at a finer granularity. Hierarchical clustering algorithms generate a clustering tree $\mathcal{T}^{(i)} = (V^{(i)}, E^{(i)}, W^{(i)})$ by iteratively merging the most similar clusters.

During the construction of the clustering tree, we use the Complete Linkage method [Defays, 1977] to measure the distance between clusters. Given two clusters $C^{(i)}_j$ and $C^{(i)}_k$ within group i, their distance is defined as:

$$\delta(C^{(i)}_j, C^{(i)}_k) = \max_{m^{(i)}_p \in C^{(i)}_j, m^{(i)}_q \in C^{(i)}_k} d(m^{(i)}_p, m^{(i)}_q) \tag{3}$$

which takes the maximum distance among all pairs of messages between the two clusters.

The hierarchical clustering algorithm proceeds as follows:

1. **Initialization.** Treat each message within group i as an independent cluster, forming the initial cluster set.
2. **Iteration.** Find the two closest clusters and merge them, while updating the clustering tree.
3. **Termination.** The algorithm terminates when there is only one cluster left in the cluster set.

To obtain the final clustering result, we need to cut the clustering tree. A common strategy is to set a distance threshold $\theta^{(i)}$, and stop merging when the weight of an edge exceeds $\theta^{(i)}$. The choice of threshold $\theta^{(i)}$ affects the granularity of clustering.

To intuitively evaluate the quality of clustering results, we introduce Silhouette Coefficient [Rousseeuw, 1987] as an unsupervised evaluation metric. Given the clustering result $\mathcal{C}^{(i)} = \{C^{(i)}_1, C^{(i)}_2, \ldots, C^{(i)}_{k_i}\}$ for group i, the Silhouette Coefficient of data point $m^{(i)}_j$ is defined as:

$$s(m^{(i)}_j) = \frac{b(m^{(i)}_j) - a(m^{(i)}_j)}{\max\{a(m^{(i)}_j), b(m^{(i)}_j)\}} \tag{4}$$

where $a(m^{(i)}_j)$ represents the average distance between $m^{(i)}_j$ and other points within the same cluster, and $b(m^{(i)}_j)$ represents the minimum average distance between $m^{(i)}_j$ and points in other clusters.

We can calculate the average value of Silhouette Coefficients for all data points within group i:

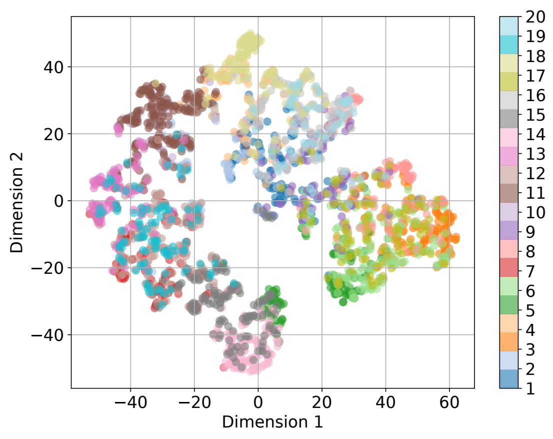

Figure 2: t-SNE visualization for the largest 20 clusters.

$$\bar{s}^{(i)} = \frac{1}{n_i} \sum_{j=1}^{n_i} s(m_j^{(i)}) \quad (5)$$

The average Silhouette Coefficient $\bar{s}^{(i)}$ reflects the overall quality of the clustering result for group i and can be used to compare the performance of different clustering algorithms or parameter settings.

Figure 2 shows the result of sampling and visualization of Bing Copilot's chat data on a workday. It can be seen from the figure that different clusters form several compact clusters in the low-dimensional space, with high semantic similarity within clusters and large semantic differences between clusters. We calculated the average Silhouette Coefficient of these 20 clusters to be 0.713, further verifying the high quality of the clustering results.

3.5 Regular Expression Extraction

From the clustering results, we need to extract regular expressions that can match most messages. We designed a difference comparison algorithm LCS-Diff based on the Longest Common Substring (LCS), which automatically generates regular expressions that match the cluster by comparing the differences between messages within the cluster. Algorithm 1 shows the complete process of regular expression extraction, where n is the size of cluster C, l is the average length of strings, k is the number of regular expression clusters.

3.6 Matching Optimization

To achieve real-time robot detection, we store the extracted regular expressions in a template database. We designed a matching algorithm based on inverted indexing and regular expression optimization to efficiently match new messages with a large number of regular expressions.

Inverted Index Design. We first build an inverted index for the keywords (ordinary strings) of each regular expression. Keywords can be generated by word segmentation algorithms. Given a new message M, we quickly find the candidate regular expression set $\mathcal{R}_M$ through the inverted index. This step can avoid the expensive overhead of matching with all regular expressions, significantly improving matching efficiency.

Algorithm 1 Regular rxpression extraction

```
Require: Cluster $C = \{s_1, s_2, \dots, s_n\}$
Ensure: Regular expression $r$
 1: $\mathcal{R} \leftarrow \emptyset$
 2: for $i \leftarrow 1$ to $n-1$ do
 3:     diff $\leftarrow$ LCS-Diff$(s_i, s_{i+1})$
 4:     $r_i \leftarrow \varepsilon$
 5:     for $(op, c, p, l, t)$ in diff do
 6:         if $t = 0$ then
 7:             $r_i \leftarrow r_i \circ c$
 8:         else
 9:             $r_i \leftarrow r_i \circ$ '.*'
10:         end if
11:     end for
12:     $r_i \leftarrow$ Optimize$(r_i, \lambda)$
13:     $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_i\}$
14: end for
15: $\{R_1, R_2, \dots, R_k\} \leftarrow$ Cluster$(\mathcal{R}, \tau)$
16: $r \leftarrow \varepsilon$
17: for $j \leftarrow 1$ to $k$ do
18:     $r_j \leftarrow$ MSA$(R_j)$.replace( ', .*')
19:     $r \leftarrow r \circ r_j \circ$ —'
20: end for
21: $r \leftarrow r[: -1]$
22: return $r$
```

Algorithm 2 Optimized regex matching

```
Require: Message $M$, inverted index $\mathcal{I}$, regular expression set $\mathcal{R}$
Ensure: Matching result $\mathcal{R}_{\text{match}}$
 1: $\mathcal{K}_M \leftarrow$ ExtractKeywords$(M)$
 2: $\mathcal{R}_M \leftarrow \emptyset$
 3: for $k \in \mathcal{K}_M$ do
 4:     $\mathcal{R}_M \leftarrow \mathcal{R}_M \cup \mathcal{I}(k)$
 5: end for
 6: $\mathcal{R}_M \leftarrow$ SortByHitRate$(\mathcal{R}_M)$
 7: $\mathcal{R}_M \leftarrow$ SortByLength$(\mathcal{R}_M)$
 8: $\mathcal{R}_{\text{match}} \leftarrow \emptyset$
 9: for $R_i \in \mathcal{R}_M$ do
10:     if Match$(M, R_i)$ then
11:         $\mathcal{R}_{\text{match}} \leftarrow \mathcal{R}_{\text{match}} \cup \{R_i\}$
12:     end if
13: end for
14: return $\mathcal{R}_{\text{match}}$
```

Regular Expression Optimization. To further accelerate the matching process, we executed a series of optimization strategies on the candidate regular expression set $\mathcal{R}_M$, such as sorting according to historical hit rate, prioritizing matching shorter expressions, structurally preprocessing and indexing regular expressions, etc.

After optimization, we obtained a refined and efficient regular expression matching workflow. Algorithm 2 shows the optimized matching process.

Finally, we comprehensively calculate the risk score Risk(M) of the message according to the matching results:

$$\text{Risk}(M) = \sum_{R_i \in \mathcal{R}_M} w_i \cdot \text{Match}(M, R_i) \quad (6)$$

where w_i is the risk weight of regular expression R_i. When Risk$(M) \geq 1$, we can directly execute an interception operation, i.e., refuse to provide service, to quickly filter out high-risk messages and improve the system's security.

4 Experiments

4.1 Dataset

We obtained real chat log data of Bing Copilot on a certain day in March 2024 from the Bing platform for analysis in this study. We randomly sampled 118,000 messages and labeled each message as either bot or human using Bing's existing hundreds of mature detection rules. These detection rules include various types such as fake User-Agent detection, IP reputation, mouse trajectory, JavaScript fingerprinting, and JA3 fingerprinting. Ultimately, we identified 33,294 bot-generated messages out of the 118,000 messages, accounting for 28.2% of the total. During the preprocessing stage, we grouped all messages using user fingerprint features. Table 1 shows the top 10 most frequent regex templates extracted from the dataset. It can be seen that these templates cover various types, including translation, text rewriting, e-commerce promotion, etc. At the same time, most templates have an average intra-class distance (Distance) less than 0.1, indicating that the internal consistency of the corresponding message clusters is high, and the extracted regular expressions can effectively summarize these messages.

We observed some typical prompt templates, as shown in Figure 3. These two cases demonstrate an Amazon seller-based promotional bot and a history event-based Q&A bot, respectively. It can be seen that although there are certain differences in the details within each cluster of messages, they all follow obvious structured patterns. LPDetective can automatically extract highly generalized regular expressions from these similar texts, thereby effectively identifying the corresponding types of bot messages.

We evaluated the performance of LPDetective on three datasets: Bing Copilot, WildChat [Zhao *et al.*, 2024], and ChatLog [Tu *et al.*, 2023]. Among them, Bing Copilot contains real conversation data from Bing AI, WildChat selects 15,783 ChatGPT interaction data, and ChatLog extracts 28,412 conversations. We manually annotated the prompt abuse in WildChat and ChatLog and found that bot-generated messages accounted for 15.8% and 14.0%, respectively. The annotation work was completed by 27 experienced engineers, all with more than 2 years of relevant work experience. Each message was independently annotated by 3 engineers to ensure accuracy and consistency. Fleiss' Kappa [Fleiss, 1971] evaluation showed high consistency among annotators, with a Kappa value of 0.93. The annotators were mainly 25-35 years old, with a bachelor's degree or above in computer-related majors and more than 2 years of relevant work experience. In this paper, these final labels are assumed to be accurate benchmark ground truth. We randomly divided each website's dataset into training set (70%), validation set (10%), and test set (20%). We ran each experiment 10 times and reported the average and standard deviation of the metrics. The standard deviation of all experiments was guaranteed to be within 0.5%. Unless otherwise specified, all experimental results reported in this section were obtained using this method. All experiments were conducted on an Ubuntu 20.04 server equipped with an Intel Xeon 8369B CPU, 96 GB memory, and an NVIDIA V100 GPU. The experimental code was implemented based on PyTorch 2.2.0.

Example 1: Amazon seller promotion bot

Message 1: When you are an amazon seller.You plan to run a cpc campaign for product:Bingo G
Message 2: When you are an amazon seller.You plan to run a cpc campaign for product:Pet U S
Message 3: When you are an amazon seller.You plan to run a cpc campaign for product:Lucky C
Extracted regex: When you are an amazon seller.You plan to run a cpc campaign for product:.*

Example 2: Verb query bot

Message 1: 10 phrasal verbs with N different from the above searched
Message 2: 17 phrasal verbs with Q different from the above searched
Message 3: 10 phrasal verbs with K different from the above searched
Extracted regex: .* phrasal verbs with .* different from the above searched*

Figure 3: Examples of typical prompt templates extracted by LPDetective.

4.2 Baseline Comparison

We implemented several baseline models, including: LSTM [Hochreiter, 1997], N-gram [Cavnar *et al.*, 1994], FastText [Joulin *et al.*, 2016], TextCNN [Kim, 2014], XLNet [Yang *et al.*, 2019], DRNN [Wang, 2018], and HiAGM [Zhou *et al.*, 2020]. All models used the Adam optimizer, and we searched for the learning rate initial value between 0.0001 and 0.1, the batch size between 16 and 128, and the number of training iterations between 10 and 1000. We selected the hyperparameter combination with the highest F1 value on the validation set as the final setting. The learning rate initial value for all models was 0.001, the batch size was 64, and the number of training iterations was 100.

Table 2 shows the performance comparison of LPDetective with baseline methods on the three datasets. Traditional machine learning methods such as N-gram have an F1 value not exceeding 0.70. Deep learning-based methods such as LSTM, FastText, TextCNN, etc. have improved F1 values, but they are still not sufficient to effectively characterize the prompt text features, and their inference latency is high. XLNet outperforms other methods in F1 value, but its inference latency is as high as tens of seconds. The F1 values of DRNN and HiAGM increase to around 0.80, but inference speed is still a bottleneck. LPDetective achieves F1 values as high as 0.940, 0.931, and 0.947 on the three datasets, respectively, with Latency controlled within 0.3 seconds. We used

$C_{\mathcal{R}}$	$Dist_{\mathcal{R}}$	$L_{\mathcal{R}}$	$\mathcal{R}$
1319	0.0629	59	.*] example sentences, translate the example sentences into modern Chinese. Return JSON, with keys: "example" and "example_explanat.*
448	0.0683	36	Answer the number I send: .*, only reply with number, don't answer anything else, be concise
171	0.0781	75	.*\Rewrite This Article in simple english language with unique keywords:n.*
117	0.0744	75	shorten the paragraph below without losing important information!nn\.*
97	0.0306	57	.*#) Reply in Chinese n This image contains fabric, please help me describe the fabric in detail, including pattern, color, etc. n Not needed
77	0.0789	75	When you are an amazon seller.You plan to run a cpc campaign for product:.*
65	0.0645	77	Please generate a list of 30 companies in China related to the .* industry. Output the results in json format, including the following json keys: company_id.*
51	0.0816	68	.*nNot answered, Scored 1.00, Not flagged, Flag, Paragraph question.*
47	0.0641	66	.*ive answer in points*20 points*+conclusion.*+way forward.*
44	0.0846	66	.*], please generate an "attractive introduction that makes people who have never been there want to go". n. Word count should be under 200 characters and in one .*

$C_{\mathcal{R}}$: Number of matched messages; $Dist_{\mathcal{R}}$: Intra-cluster average distance; $L_{\mathcal{R}}$: Regex length; $\mathcal{R}$: Regular expression

Table 1: Top 10 most frequent regex templates, sorted by the number of matched messages

the Wilcoxon signed-rank test to compare the differences in F1 values between LPDetective and each baseline method at a significance level of p¡0.01, and the results showed that the superiority of LPDetective was statistically significant (p¡0.01), rather than random noise. The excellent performance of LPDetective is attributed to: 1. Automatically mining key patterns from massive real data to construct a highly structured and fine-grained prompt template database; 2. Systematic exploration in matching optimization and introduction of a risk scoring mechanism to balance precision and recall.

4.3 Prompt Language Analysis

We sorted the regular expressions extracted from the dataset in descending order of length. Table 3 shows the distribution of regular expressions in various languages. It can be seen that English is the most widely used language for prompt templates, accounting for 40.84%. For languages with larger character lengths such as Chinese and Japanese, the P_1 length of their regular expressions is significantly higher than other languages, reaching 142, 145, and 148, respectively. In contrast, for languages such as Indonesian and Portuguese, the P_1 length of their regular expressions is relatively shorter, at 117 and 122, respectively. This indicates that bots in these regions tend to use shorter prompt templates.

4.4 Prompt Topic Analysis

We used the GPT-4 model to classify the content topics of the extracted regular expressions. We divided the regular expressions into 9 topic categories, as shown in Table 4. It can be seen that Programming is a key area of bot applications, accounting for 8.40%. This indicates that some developers are trying to leverage Bing Copilot's code generation capabilities to assist with programming or debugging tasks. The proportion of the Math Problems topic is also relatively high, reaching 5.73%, and the P_1 value of the regular expression length is as high as 162. This may be due to the complexity of mathematical problems themselves, resulting in a relatively large length of the corresponding regular expressions. Additionally, we found that 54.19% of the regular expressions were difficult to classify into existing topics and were marked as the Others category. We also discovered that some regular expressions involved pornographic and gambling content, highlighting the necessity of strengthening content moderation.

4.5 Ablation Study

We conducted ablation experiments on three datasets to examine the following variants:

- **w/o Grouping.** Remove the message grouping step and directly cluster and extract regular expressions.
- **w/o Clustering.** Remove the hierarchical clustering step and directly compare messages within each group to extract regular expressions.
- **w/o Optimization.** Do not optimize during regular expression extraction and retain all non-wildcard parts.
- **w/o Index.** Do not use inverted index during matching and match with all regular expressions.

Table 5 shows the results of the ablation experiments. Removing message grouping (w/o Grouping) reduces the F1 value by 0.085, indicating that pre-clustering using user fingerprints can improve the cohesion of prompt templates. Removing hierarchical clustering (w/o Clustering) leads to a decrease of 0.124 in F1, indicating that two-stage clustering can effectively organize similar messages and facilitate the extraction of more accurate regular expressions. Removing regular expression extraction optimization (w/o Optimization) generates lengthy expressions, reduces interpretability, and increases detection latency by 3.628 seconds. Removing the inverted index (w/o Index) increases the detection latency from 0.294 seconds to 8.947 seconds.

We also analyzed the impact of key parameters: the minimum non-wildcard substring length threshold L and the clustering stop distance threshold θ on LPDetective. Table 6 shows that increasing L makes regular expressions more concise, but when L exceeds 5, overly simple expressions instead cause the F1 value to decrease, dropping to 0.820 at $L = 10$. This indicates that excessive simplification of regular expressions loses important pattern information and leads

Method	Bing Copilot				Wildchat				ChatLog			
	Prec.	Rec.	F1	Latency	Prec.	Rec.	F1	Latency	Prec.	Rec.	F1	Latency
LSTM	0.762	0.742	0.752	15.283	0.749	0.728	0.738	18.419	0.756	0.736	0.746	16.852
N-gram	0.698	0.676	0.687	2.617	0.674	0.652	0.663	3.140	0.686	0.664	0.675	2.879
FastText	0.735	0.715	0.725	1.536	0.718	0.698	0.708	1.844	0.729	0.709	0.719	1.690
TextCNN	0.793	0.775	0.784	5.201	0.772	0.752	0.762	6.242	0.783	0.763	0.773	5.722
XLNet	0.873	0.857	0.865	52.438	0.860	0.842	0.851	62.927	0.867	0.851	0.859	57.683
DRNN	0.807	0.789	0.798	12.456	0.795	0.775	0.785	14.948	0.802	0.784	0.793	13.702
HiAGM	0.835	0.819	0.827	10.175	0.821	0.803	0.812	12.211	0.828	0.810	0.819	11.193
LPDetective	**0.946**	**0.934**	**0.940**	**0.294**	**0.938**	**0.924**	**0.931**	**0.253**	**0.952**	**0.942**	**0.947**	**0.265**

Prec.: Precision, Rec.: Recall, F1: F1-score, Latency: Processing time in seconds per 1,000 messages

Table 2: Performance of different prompt abuse detection methods on datasets: Bing Copilot, Wildchat, and ChatLog

Language	Regex Length			Percentage
	P_1	P_{50}	P_{99}	
English	135	62	28	40.84%
Spanish	128	58	25	7.25%
Indonesian	117	51	22	5.73%
Chinese	142	68	31	5.34%
Italian	131	60	27	4.58%
Japanese	145	71	33	4.20%
Vietnamese	127	57	24	3.82%
Portuguese	122	54	23	3.44%
Others	133	61	27	24.80%

Table 3: Prompt template languages

Topic	Regex Length			Percentage
	P_1	P_{50}	P_{99}	
Programming	138	64	29	8.40%
Medicine & Health	131	60	27	7.25%
Education & Exams	134	62	28	6.87%
Math Problems	162	88	41	5.73%
Daily Life	119	52	23	4.96%
Travel	123	55	24	4.58%
Social Chat	140	66	30	4.20%
Sports & Fitness	133	61	27	3.82%
Others	132	61	27	54.19%

Table 4: Prompt template topics

Method	F1	Latency (s/1k messages)
LPDetective	**0.940**	**0.294**
w/o Grouping	0.855	0.372
w/o Clustering	0.816	0.453
w/o Optimization	0.902	3.922
w/o Index	0.940	8.947

Table 5: Ablation analysis for LPDetective

(a) $\theta = 0.05$				(b) $L = 5$			
L	F1	L	F1	θ	F1	θ	F1
1	0.852	6	0.903	0.01	0.943	0.12	0.865
2	0.893	7	0.874	0.03	0.941	0.15	0.841
3	0.912	8	0.852	0.05	**0.940**	0.2	0.809
4	0.928	9	0.835	0.07	0.926	0.25	0.783
5	**0.940**	10	0.820	0.1	0.887	0.3	0.762

Table 6: Parameter sensitivity analysis

to a decrease in generalization ability. θ controls the granularity of clustering, and as θ increases from 0.01 to 0.3, the F1 value generally decreases, reaching 0.762 at $\theta = 0.3$. The larger θ is, the coarser the clustering, the greater the difference within clusters, and the worse the generalization ability of the extracted regular expressions. When θ is less than 0.05, the clustering is too fragmented, slightly improving the F1 value but significantly reducing detection efficiency. Reasonable selection of L and θ is crucial for LPDetective, and experiments show that $L = 5$ and $\theta = 0.05$ is the optimal parameter combination.

5 Discussion

Although LPDetective demonstrated superior performance in our experiments, it still has some limitations. First, LPDetective mainly relies on prompt templates mined from historical conversation data, and it may be difficult to identify new, unseen abnormal conversation patterns in a timely manner. Second, LPDetective has been primarily optimized for English conversation data, and its adaptability to other languages needs further verification. Third, LPDetective needs to match a large number of regular expressions during inference. Although we have reduced the matching overhead through optimization measures such as inverted indexes, the inference latency may still fail to meet real-time requirements in scenarios with an extremely large number of templates.

6 Conclusion

To address the threat of bot abuse faced by large language model-driven dialogue systems, this paper proposes LPDetective, a novel unsupervised detection framework. Our large-scale experiments on three real-world datasets, Bing Copilot, Wildchat, and ChatLog, show that LPDetective significantly outperforms existing methods in terms of accuracy, recall, and real-time performance. For example, on the Bing Copilot dataset, LPDetective achieves an F1 score of 94.0%, an improvement of 7.5% over the best baseline XLNet, with a detection latency of only 0.294 seconds, which is 1/178 of XLNet. In the future, we plan to explore combining few-shot learning, multilingual optimization, hierarchical indexing, and other techniques to enhance its practicality.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant No. 2022YFB2703301.

Ethical Impact

During the development of LPDetective, we placed great emphasis on protecting user privacy and data security. We ensure that all data used for analysis has been desensitized and anonymized.

References

[Agomuoh, 2022] F. Agomuoh. Your siri conversations may have been recorded without your permission. https://www.digitaltrends.com/computing/developer-finds-apple-bluetooth-security-flaw-worth-7000/, 2022.

[Ayanouz *et al.*, 2020] Soufyane Ayanouz, Boudhir Anouar Abdelhakim, and Mohammed Benhmed. A smart chatbot architecture based nlp and machine learning for health care assistance. In *Proceedings of the 3rd international conference on networking, information systems & security*, pages 1–6, 2020.

[Baki *et al.*, 2017] Shahryar Baki, Rakesh Verma, Arjun Mukherjee, and Omprakash Gnawali. Scaling and effectiveness of email masquerade attacks: Exploiting natural language generation. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 469–482, 2017.

[Cavnar *et al.*, 1994] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175, page 14. Las Vegas, NV, 1994.

[Chung *et al.*, 2017] Hyunji Chung, Michaela Iorga, Jeffrey Voas, and Sangjin Lee. Alexa, can i trust you? *Computer*, 50(9):100–104, 2017.

[CNBC, 2023] CNBC. Openai says chatgpt downtime caused by targeted attack, 2023.

[Coles, 2023] C Coles. 11% of data employees paste into chatgpt is confidential. *Cyberhaven. Accessed: May*, 1, 2023.

[Defays, 1977] Daniel Defays. An efficient algorithm for a complete link method. *The computer journal*, 20(4):364–366, 1977.

[Economist, 2023] T. Economist. Concerns turned into reality... as soon as samsung electronics unlocks chatgpt, 'misuse' continues. https://economist.co.kr/article/view/ecn202303300057?s=31, 2023.

[Fleiss, 1971] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[Følstad and Skjuve, 2019] Asbjørn Følstad and Marita Skjuve. Chatbots for customer service: user experience and motivation. In *Proceedings of the 1st international conference on conversational user interfaces*, pages 1–9, 2019.

[Frieder *et al.*, 2023] S. Frieder, L. Pinchetti, R.-R. Griffiths, T. Salvatori, T. Lukasiewicz, P. C. Petersen, A. Chevalier, and J. Berner. Mathematical capabilities of chatgpt. *arXiv preprint arXiv:2301.13867*, 2023.

[Giaretta and Dragoni, 2020] Alberto Giaretta and Nicola Dragoni. Community targeted phishing: A middle ground between massive and spear phishing through natural language generation. In *Proceedings of 6th International Conference in Software Engineering for Defence Applications: SEDA 2018 6*, pages 86–93. Springer, 2020.

[Gondaliya *et al.*, 2020] Krishna Gondaliya, Sergey Butakov, and Pavol Zavarsky. Sla as a mechanism to manage risks related to chatbot services. In *2020 IEEE 6th 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)*, pages 235–240. IEEE, 2020.

[Gupta *et al.*, 2020] Aishwarya Gupta, Divya Hathwar, and Anupama Vijayakumar. Introduction to ai chatbots. *International Journal of Engineering Research and Technology*, 9(7):255–258, 2020.

[Hochreiter, 1997] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[Hurler, 2023] K. Hurler. Amazon warns employees to beware of chatgpt. https://gizmodo.com/amazon-chatgpt-ai-software-job-coding-1850034383, 2023.

[Hussain *et al.*, 2019] Shafquat Hussain, Omid Ameri Sianaki, and Nedal Ababneh. A survey on conversational agents/chatbots classification and design techniques. In *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*, pages 946–956. Springer, 2019.

[Integration-Automation, 2023] Integration-Automation. Reedgegpt - the reverse engineering the chat feature of the new version of bing, 2023.

[Joulin *et al.*, 2016] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[Karanjai, 2022] R. Karanjai. Targeted phishing campaigns using large scale language models. *arXiv preprint arXiv:2301.00665*, 2022.

[Kim, 2014] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[Lei *et al.*, 2017] X. Lei, G.-H. Tu, A. X. Liu, K. Ali, C.-Y. Li, and T. Xie. The insecurity of home digital voice assistants–amazon alexa as a case study. *arXiv preprint arXiv:1712.03327*, 2017.

[Levenshtein, 1966] VI Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Proceedings of the Soviet physics doklady*, 1966.

[McGee, 2023] Robert W McGee. What will the united states look like in 2050? a chatgpt short story. *A Chatgpt Short Story (April 8, 2023)*, 2023.

[Mijwil and Aljanabi, 2023] M. Mijwil and M. Aljanabi. Towards artificial intelligence-based cybersecurity: The practices and chatgpt generated ways to combat cybercrime. *Iraqi Journal For Computer Science and Mathematics*, 4:65–70, 2023.

[Mulgrew, 2023] Aaron Mulgrew. I built a zero day virus with undetectable exfiltration using only chatgpt prompts. *Forcepoint. Retrieved May*, 17:2023, 2023.

[OpenAI, 2023] OpenAI. Gpt-4 technical report. https://arxiv.org/abs/2303.08774, 2023.

[Ouyang *et al.*, 2022] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, and A. Ray. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022.

[OWASP, 2023] OWASP. OWASP top 10 for LLM, 2023.

[Paliwal *et al.*, 2020] Shweta Paliwal, Vishal Bharti, and Amit Kumar Mishra. Ai chatbots: Transforming the digital world. *Recent trends and advances in artificial intelligence and internet of things*, pages 455–482, 2020.

[Point, 2023] Check Point. Opwnai: Cybercriminals starting to use chatgpt. *Check Point. Retrieved May*, 15:2023, 2023.

[Prakken, 2020] H. Prakken. A persuasive chatbot using a crowd-sourced argument graph and concerns. *Computational Models of Argument*, 326:9, 2020.

[Qian *et al.*, 2023] C. Qian, H. Qi, G. Wang, L. Kunc, and S. Potdar. Distinguish sense from nonsense: Out-of-scope detection for virtual assistants. *arXiv preprint arXiv:2301.06544*, 2023.

[Rousseeuw, 1987] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[Sadowski and Levin, 2007] Caitlin Sadowski and Greg Levin. Simhash: Hash-based similarity detection. In *Technical report, Google*, 2007.

[Security, 2018] P. Security. Cortana security flaw means your pc may be compromised. https://www.pandasecurity.com/mediacenter/mobile-news/cortana-security-flaw/, 2018.

[Shu *et al.*, 2020] Kai Shu, Suhang Wang, Dongwon Lee, and Huan Liu. Mining disinformation and fake news: Concepts, methods, and recent advancements. *Disinformation, misinformation, and fake news in social media: Emerging research challenges and opportunities*, pages 1–19, 2020.

[Stiff and Johansson, 2022] Harald Stiff and Fredrik Johansson. Detecting computer-generated disinformation. *International Journal of Data Science and Analytics*, 13(4):363–383, 2022.

[Tu *et al.*, 2023] Shangqing Tu, Chunyang Li, Jifan Yu, Xiaozhi Wang, Lei Hou, and Juanzi Li. Chatlog: Recording and analyzing chatgpt across time. *arXiv preprint arXiv:2304.14106*, 2023.

[Van Dis *et al.*, 2023] Eva AM Van Dis, Johan Bollen, Willem Zuidema, Robert Van Rooij, and Claudi L Bockting. Chatgpt: five priorities for research. *Nature*, 614(7947):224–226, 2023.

[VARINDIA, 2022] VARINDIA. Chatgpt produces malicious emails and code. https://varindia.com/news/chatgpt-produces-malicious-emails-and-code, 2022.

[Wang, 2018] Baoxin Wang. Disconnected recurrent neural networks for text categorization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*, pages 2311–2320, 2018.

[Wikipedia, 2022] Wikipedia. Chatgpt. https://en.wikipedia.org/wiki/ChatGPT, 2022.

[Wollny *et al.*, 2021] Sebastian Wollny, Jan Schneider, Daniele Di Mitri, Joshua Weidlich, Marc Rittberger, and Hendrik Drachsler. Are we there yet?-a systematic literature review on chatbots in education. *Frontiers in artificial intelligence*, 4:654924, 2021.

[Xu *et al.*, 2021] Jing Xu, Da Ju, Margaret Li, Y-Lan Boureau, Jason Weston, and Emily Dinan. Bot-adversarial dialogue for safe conversational agents. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2950–2968, 2021.

[Yang *et al.*, 2019] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: generalized autoregressive pretraining for language understanding. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[Zellers *et al.*, 2019] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi. Defending against neural fake news. In *Advances in neural information processing systems*, volume 32, 2019.

[Zhao *et al.*, 2024] Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. *arXiv preprint arXiv:2405.01470*, 2024.

[Zhou *et al.*, 2020] Jie Zhou, Chunping Ma, Dingkun Long, Guangwei Xu, Ning Ding, Haoyu Zhang, Pengjun Xie, and Gongshen Liu. Hierarchy-aware global model for hierarchical text classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 1106–1117, 2020.