LLM App Squatting and Cloning

Yinglin Xie*, Xinyi Hou*, Yanjie Zhao, Kai Chen† and Haoyu Wang†
Huazhong University of Science and Technology, Wuhan, China
xieyinglin@hust.edu.cn, xinyihou@hust.edu.cn, yanjie_zhao@hust.edu.cn,
kchen@hust.edu.cn, haoyuwang@hust.edu.cn

***Abstract*—Impersonation tactics, such as app squatting and app cloning, have posed longstanding challenges in mobile app stores, where malicious actors exploit the names and reputations of popular apps to deceive users. With the rapid growth of Large Language Model (LLM) stores like GPT Store and FlowGPT, these issues have similarly surfaced, threatening the integrity of the LLM app ecosystem. In this study, we present the first large-scale analysis of LLM app squatting and cloning using our custom-built tool, LLMappCrazy. LLMappCrazy covers 14 squatting generation techniques and integrates Levenshtein distance and BERT-based semantic analysis to detect cloning by analyzing app functional similarities. Using this tool, we generated variations of the top 1000 app names and found over 5,000 squatting apps in the dataset. Additionally, we observed 3,509 squatting apps and 9,575 cloning cases across six major platforms. After sampling, we find that 18.7% of the squatting apps and 4.9% of the cloning apps exhibited malicious behavior, including phishing, malware distribution, fake content dissemination, and aggressive ad injection.**

I. Introduction

Mobile app squatting [19], where attackers publish apps with identifiers (e.g., app or package names) that mimic popular apps, such as through typosquatting (e.g., changing "Facebook" to "Fecebook"), is a growing threat in the mobile ecosystem. Hu *et al.*[19] identified over 10,553 squatting apps targeting the top 500 apps on Google Play, with more than 51% classified as malicious and some reaching millions of downloads. These counterfeit apps pose serious risks, including data theft and malware infections. Despite mitigation efforts by platforms, the sheer number of apps and sophisticated squatting tactics make detection and prevention difficult.

Inspired by the extensive research on mobile app squatting, we have turned our attention to similar threats within emerging Large Language Model (LLM) app stores [45]. With the rise of LLMs, such as ChatGPT [27], Gemini [14], and Claude [10], there has been a proliferation of applications that leverage these models in diverse domains, including chatbots, content generation tools, and virtual assistants [6], [9], [11], [13], [28], [29]. LLM-powered applications have gained immense popularity due to their ability to perform complex tasks, leading to the creation of entire app ecosystems around them. However, as these LLM app stores continue to expand rapidly, we observe that they are becoming fertile ground for **LLM app squatting** attacks similar to those in traditional mobile app markets, as shown in Figure 1. In this context, squatting primarily occurs at the app identifier level, where attackers create apps with names that closely mimic legitimate ones to deceive users. For example, squatting could manifest as subtle name changes or the addition of enticing words, such as "Canva Pro", tricking users into believing they are using an official or enhanced version of a popular app. Moreover, LLM app stores have significantly lowered the barrier to entry for developers. This democratization of development allows individuals from various backgrounds, even those with limited programming experience, to create and publish apps. While this inclusivity fosters innovation, it also makes it easier for attackers to clone the entire LLM app not only the app's name but also its functionality and behavior. We refer to this more insidious form of attack as **LLM app cloning**, where the cloned app mirrors the legitimate one in nearly every aspect, making it even harder for users to discern the difference.

To comprehensively investigate squatting and cloning in LLM app stores, we focus on six prominent LLM app stores (i.e., GPT Store [28], FlowGPT [13], Poe [29], Coze [11], Cici [9], and Character.AI [6]) that have gained significant traction due to the widespread adoption of LLM-powered applications. In our study, we develop a tool, LLMappCrazy, designed to automatically detect squatting and cloning instances within these ecosystems. Using LLMappCrazy, we systematically examine app identifier variations and functional cloning across GPT Store, identifying potential 5,834 name squatting apps and 6094 name cloning apps. And we also detect other features of apps in six LLM app stores. Our results reveal the scope of the problem: we found 3,509 squatting apps, and 9,575 cloned apps, confirming that this phenomenon is not isolated to mobile app markets but is rapidly spreading into the LLM domain. The findings indicate that 18.7% of the squatting apps and 4.9% of cloning apps exhibited malicious behavior, and some of them had amassed significant user downloads, further exacerbating the security risks faced by users of LLM-based applications.

Contributions. We make the following main contributions:

1) To the best of our knowledge, this is the first detailed investigation into squatting and cloning attacks within LLM app stores.
2) We develop LLMappCrazy, a tool that detects squatting and cloning apps using 14 squatting-generation techniques and advanced semantic analysis.
3) Using LLMappCrazy, we find 5,834 name squatting apps and 6094 name cloning apps; We conduct a large-scale

*Yinglin Xie and Xinyi Hou are the co-first authors.
†Corresponding authors.

empirical study across six LLM app stores, identifying 3,509 squatting apps, and 9,575 cloning apps.

4) We find that 18.7% of the identified squatting apps and 4.9% of cloning apps exhibit malicious behavior, including phishing, malware, and ad injection. And we identify 227 apps that exhibit a high degree of similarity in various features to other apps in GPT Store.
5) We study the impact of LLM app squatting and cloning, discovering that these apps have reached up to 2.7 million conversations, posing significant risks to platform trust.

II. Background and Related Work

A. LLM App Store

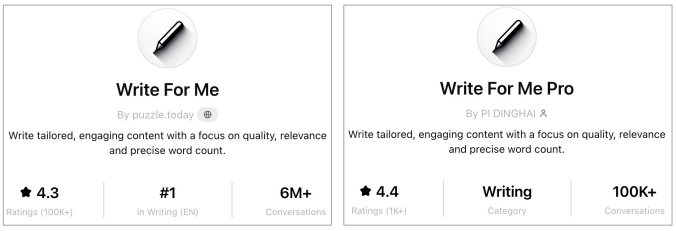
LLMs are advanced AI systems designed to understand and generate human language. Trained on vast datasets, they produce coherent, contextually relevant responses to a wide range of prompts. As LLM technology has progressed, **LLM apps** [45] have emerged. These are software applications powered by LLMs, designed to perform specific tasks such as text generation, translation, and conversational interactions. At the same time, **LLM app stores** act as centralized platforms for discovering, distributing, and managing these apps. Platforms like OpenAI's GPT Store [28] have become key hubs for users and developers to access and share LLM apps.

Several studies explored the ecosystem and security of LLM app stores. Zhao *et al.* [45] provided a vision and roadmap for the analysis of LLM app stores, outlining the future directions for research. Zhang *et al.* [44] conducted an initial analysis of GPTs distribution and potential vulnerabilities, while Su *et al.* [33] provided comprehensive mining of the GPT Store, examining app characteristics and user engagement. Additionally, Yan *et al.* [41] explored the GPT Store ecosystem, focusing on distribution, deployment, and security aspects. To support further research, Hou *et al.* [18] introduced GPTZoo, a dataset containing over 730,000 GPT instances. In terms of security, Hou *et al.* [17] examined the security of LLM app stores, highlighting critical vulnerabilities and security challenges in these platforms. Tao *et al.* [35] discussed the risks associated with custom GPTs, Hui *et al.* [21] uncovered vulnerabilities related to prompt leaking attacks. Antebi *et al.* [5], [24] analyzed the misuse of custom GPTs and malicious services integrated with LLMs, respectively.

However, while these works cover various aspects of LLM apps, the issues of LLM app impersonation, such as squatting and cloning, remain underexplored. These emerging threats pose significant risks to the expanding LLM app ecosystem and warrant further investigation.

B. Squatting Attack.

Domain squatting[40] involves registering domains similar to legitimate ones with malicious intent. A common form, *typosquatting*, exploits users' typographical errors when typing domain names, diverting traffic from legitimate sites. Agten*et al.* [3], [32] provide detailed analyses of typosquatting, with the latter highlighting the effectiveness of character permutations and substitutions in deceiving users.



(a) A popular LLM app. (b) A squatting app.

Fig. 1: An example of LLM app squatting.

Domain squatting was traditionally linked to web attacks but has since expanded into other areas. Szurdi *et al.* [34] examined *email typosquatting*, where attackers register emails similar to legitimate ones to intercept communications or conduct phishing. Griffiths [16] explored its role in business email compromise (BEC) attacks. Squatting has also spread to programming package managers, where attackers publish malicious packages with names resembling popular libraries, as seen in *package typosquatting* in PyPI, RubyGems, and NPM [36], [37], [38]. Taylor *et al.* [37] suggested defense strategies, while Vu *et al.* [38] analyzed typosquatting in Python. In the mobile app ecosystem, Hu *et al.* [19] investigated *mobile app squatting*, where malicious apps use names similar to legitimate ones to deceive users. Chen *et al.* [8] introduced *GUI-squatting*, where phishing apps replicate the graphical interface of legitimate apps to trick users into providing sensitive information.

While squatting in traditional domains, emails, package managers, and mobile apps have been extensively studied, squatting within LLM app stores remains an underexplored area. Our work seeks to fill this gap.

C. Cloning Detection

Cloning has been widely studied in software development, especially in mobile app ecosystems, where cloned apps raise significant security concerns such as malware distribution, intellectual property theft, and privacy violations. Rattan *et al.* [30] reviewed software clone detection, highlighting challenges like bug propagation and maintenance issues. In mobile apps, various studies have focused on detecting clones in both official and unofficial markets. Crussell *et al.* [12] first addressed the issue with detection methods based on app metadata and code similarity. Wang *et al.* [39] introduced *Wukong*, a scalable two-phase approach using static and dynamic analysis. Chen *et al.* [7] proposed a hybrid method balancing accuracy and scalability, while Lyu *et al.* [25] developed SuiDroid, a system resilient to obfuscation. Niu *et al.* [26] combined static and dynamic analysis for clone detection, and Hu *et al.* [20] introduced a UI-based approach to detect clones mimicking the visual design of legitimate apps.

Recent advancements in clone detection have utilized machine learning and deep learning models. Zhang *et al.* [43] highlighted the vulnerabilities of machine learning-based detectors when faced with semantic-preserving code transfor-

mations, showing how subtle syntax changes can bypass detection. Khajezade *et al.* [22] evaluated few-shot and contrastive learning methods, demonstrating their effectiveness in detecting clones with minimal labeled data, making them ideal for large-scale or evolving ecosystems.

As LLM app stores grow, cloning challenges are likely to arise. While advanced detection techniques like machine learning are crucial for safeguarding these stores, their effectiveness for LLM cloning remains unclear. We aim to explore this issue.

III. Motivating Study

The aforementioned research highlights the potential risks posed by LLM app squatting and cloning, indicating these threats may be widespread in the LLM app ecosystem. To explore this, we conduct a preliminary study to (1) confirm the presence of these threats and (2) assess whether existing squatting detection techniques can effectively identify them. This serves as the foundation for our later methodology.

A. Methodology

To detect potential squatting in LLM apps, we generate variations of several popular app names from the GPT Store and check for their existence in online repositories.

Generating squatting names. We begin by selecting the top 10 recommended LLM apps from the GPT Store, each with significant user engagement, as shown in Table I. For each app, we manipulate the names to create potential squatting variations that attackers could exploit. We use AppCrazy[19], a tool inspired by domain squatting generators like URLCrazy [2] and DNSTwist [1]. AppCrazy includes 11 models tailored for mobile app ecosystems, such as punctuation deletion (e.g., "DALL·E" to "DALLE"), character insertion (e.g., "DALL·E" to "DALLLEE"), and substitution (e.g., "DALL·E" to "DALL3"). Using these models, we generate 625 variations from the app names of the 10 selected apps.

Verifying squatting names. To verify whether these squatting names exist in the wild, we rely on GPTZoo [18], a metadata dataset that tracks over 730,000 LLM apps from the GPT Store. We run an automated search using the 625 generated squatting names in the GPTZoo dataset. This search returns 32 results that match our squatting name variations. We then manually verify these apps using the GPT Store to determine whether the apps are legitimate or potential squatting attempts. This manual review is crucial for eliminating false positives. Through this process, we identify 28 apps that appear to be squatting on popular LLM app names, demonstrating the prevalence of squatting in the LLM app ecosystem.

B. Motivating Results

As shown in Table I, we identified 28 squatting apps. Of the 11 generation models used by AppCrazy, only 4 proved effective in generating squatting apps. Out of the 625 name strings generated, only 28 matched real squatting apps, meaning that more than 95.52% of the generated strings did not identify any squatting cases. Interestingly, during this process, we also encountered squatting apps not directly identified by the names generated by AppCrazy. For instance, when querying the GPTZoo dataset, we found several "related" apps. Manually reviewing these results, we identified 34 squatting apps that did not directly match the names generated by AppCrazy.

TABLE I: Results of the motivating study.

App Name (# Chats)	AppCrazy		LLMappCrazy	
	# Generated	# Identified	# Generated	# Identified
Image Generator (6M+)	78	2	460	36
Consensus (5M+)	39	0	419	0
Write For Me (4M+)	53	1	447	4
Logo Creator (2M+)	59	9	441	33
Canva (2M+)	22	1	403	2
Scholar GPT (2M+)	29	7	412	15
Code Copilot (2M+)	65	7	447	8
Cartoonize Yourself (2M+)	92	1	475	2
Diagrams[1](1M+)	175	0	10,623	0
Python (1M+)	13	0	393	106
Total	625	28	14,520	206

[1] Diagrams: The full name of this app is "Diagrams: Show Me | charts, presentations, code".

C. Observations

Our study confirms the existence of squatting and cloning threats in the LLM app ecosystem but also reveals significant limitations in current detection methods, including missed squatting apps and inefficiencies in name-generation models. Manual review, while effective in reducing false positives, is not scalable, emphasizing the need for improved, automated filtering techniques. Additionally, due to structural differences between LLM and traditional apps, existing cloning detection methods are inadequate, prompting the need for more tailored approaches, which we explore in the following sections.

D. Terminology

In LLM app stores, attackers often employ two primary impersonation techniques: **LLM app squatting** and **LLM app cloning**. These methods enable attackers to mislead users, either by creating apps with names similar to legitimate ones or by replicating the functionality of popular apps. Below, we define these two forms of impersonation in detail.

1) **Squatting LLM apps**: Apps that have either identical or slightly altered names to legitimate LLM apps.
2) **Cloning LLM apps**: Apps that replicate the functionality and overall user experience of legitimate LLM apps.

Squatting generation models generate potential squatting names by applying techniques like character modifications to legitimate app names. In contrast, **cloning detection models** identify cloned apps by analyzing functional similarities and detecting apps that replicate key features of legitimate ones.

IV. Approach

Our approach to identifying squatting and cloning LLM apps consists of three main steps: data collection, squatting generation, and cloning detection, as shown in Figure 2.

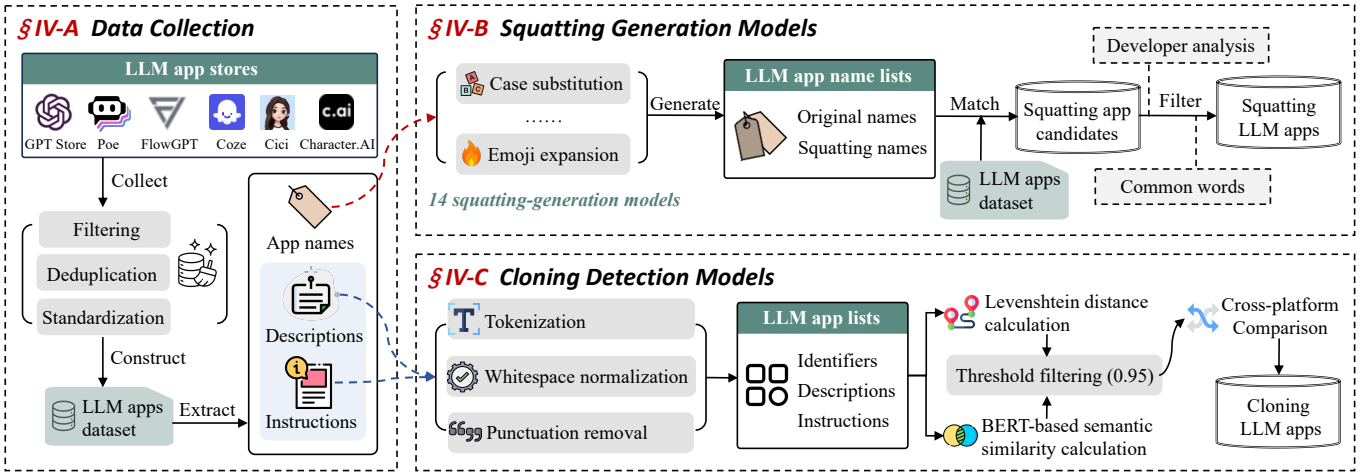


Fig. 2: Our approach to identifying squatting and cloning LLM apps.

A. Data Collection

We collected app information by scraping data from six LLM app stores: GPT Store [28], FlowGPT [13], Poe [29], Coze [11], Cici [9], and Character.AI [31]. Then, we applied several processes to ensure its accuracy and quality, including filtering, deduplication, and standardization. First, filtering was necessary because certain LLM apps might have common names not exclusive to any specific app or brand. Both the complete dataset and the filtered apps were retained and used in subsequent experiments to detect name duplication or squatting (reasons discussed § VII-B). Next, we performed deduplication by comparing app `ids`, which are unique to each app, to ensure that the dataset contained unique entries. Finally, we standardized the data into JSON format to facilitate the smooth execution of experiments and ensure reliable results. Our analysis focused on three key fields: `app name`, `description`, and `instructions`. The `app name` was used in experiments to detect duplicate or squatting names, while both the `description` and `instructions` were utilized for cloning detection, with the `description` showcasing the app's public-facing features and the `instructions` serving as its behavioral guide, similar to source code.

B. Squatting Generation Models

Inspired by the squatting name techniques introduced in AppCrazy [19], we developed LLMappCrazy, a tool tailored for detecting squatting in the emerging ecosystem of LLM apps. While LLMappCrazy builds upon the foundation of AppCrazy, our preliminary investigation revealed several key differences between mobile app squatting and LLM app squatting. To address this, we extended AppCrazy introducing methods like emoji and string expansions. Additionally, we adapted several package name squatting techniques from AppCrazy to suit LLM apps. As illustrated in Figure 3, LLMappCrazy employs 14 squatting generation models.

Mutation-based models. We retain six mutation-based models from AppCrazy, which generate squatting names by exploiting typographical errors. Below are the models we modified to address the specific characteristics of LLM apps.

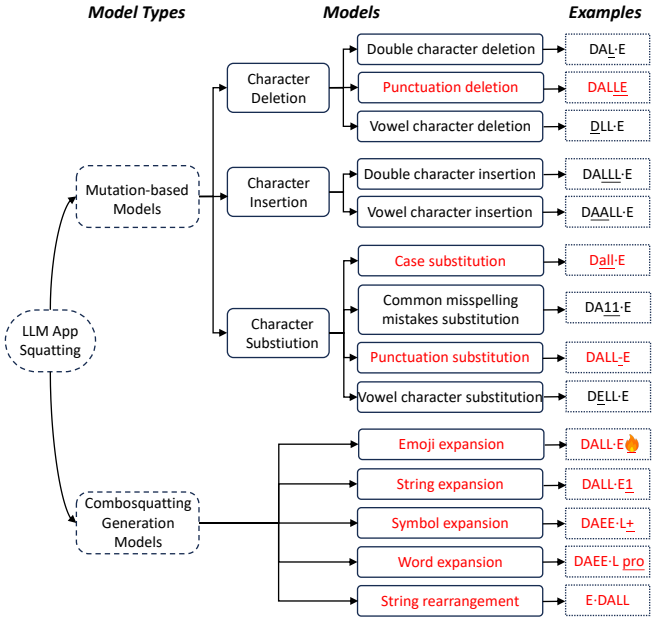


Fig. 3: The 14 kinds of LLM app squatting-generation models used in this work. The 6 models in **black** are inherited from AppCrazy [19], while the 8 models in **red** that are either newly introduced or modified in LLMappCrazy to target LLM apps.

1) *Case Substitution*: Changing uppercase characters to lowercase and vice versa, e.g., "DALL·E" into "dall·e".
2) *Punctuation Deletion*: Removing punctuation marks entirely, e.g., "DALL·E" becomes "DALLE".
3) *Punctuation Substitution*: Replacing punctuation marks with others (e.g., space, underscore), e.g., "DALL·E" becomes "DALL-E".

Combosquatting generation models. We extend the traditional combosquatting generation models to include five distinct techniques that are especially relevant to LLM apps.

In addition to the standard string manipulations, we introduce new techniques that account for the unique use of symbols and emojis in LLM app names:

1) *String Expansion*: Adding characters before or after the app name, e.g., "DALL·E" into "DALL·E1".
2) *Symbol Expansion*: Inserting or replacing characters with symbols such as "+", "#", or "$", e.g., "DALL·E" into "DALL·E+" or "DALL·E#".
3) *Word Expansion*: Appending or prepending descriptive words to the app name, e.g., "DALL·E" into "DALL·E pro" or "DALL·E AI".
4) *Emoji Expansion*: Adding emojis to the app name, e.g., "DALL·E" into "DALL·E🔥" , exploiting the visual appeal and perceived legitimacy conveyed by emojis.
5) *String Rearrangement*: Rearranging parts of the package name, e.g., "DALL·E" to "E·DALL".

Evaluation. To evaluate the effectiveness of squatting generation models, we compare the results from our tool, LLMappCrazy, with those of the traditional domain squatting approach, AppCrazy, used in the motivating study (see § III). The same set of 10 popular apps is used. With LLMappCrazy, 14,520 squatting names are generated (as shown in Columns 4-5 of Table I). Consistent with the process in the motivating study, these squatting names are searched in the GPTZoo dataset. The search identifies 206 squatting LLM app candidates with distinct IDs. Figure 4 shows the confirmed squatting apps generated by the 14 squatting-generation models. The newly added word expansion model is the most effective, with 114 squatting apps falling into this category.

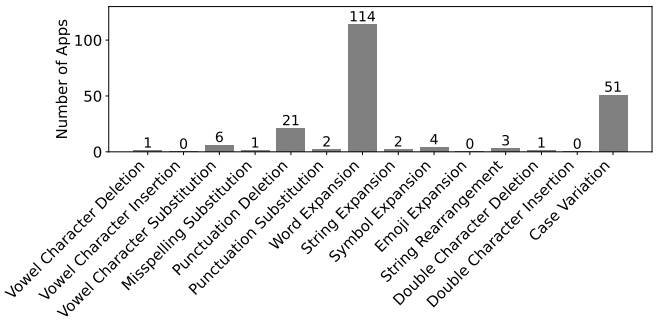


Fig. 4: The distribution of squatting apps across models.

C. Cloning Detection Models

We employed **Levenshtein distance** and **BERT-based semantic similarity** to detect plagiarism or app cloning in LLM app `descriptions` and `instructions`. Levenshtein distance identified exact or near-exact matches by measuring minimal edits, while the BERT model captured deeper semantic similarities, even with different wording. By analyzing both these components, we effectively detected cloning attempts, revealing instances of content replication ranging from direct copying to subtle paraphrasing, and highlighting the prevalence of cloning in the LLM app ecosystem.

1) Levenshtein distance calculation

To detect cases of content cloning with minor variations, we employed Levenshtein distance algorithm [42], which **calculates the minimum number of single-character edits (insertions, deletions, or substitutions)** required to transform one string into another. For each app pair, we computed the Levenshtein distance between their `instructions` fields, which act as the core content or behavioral guide of the LLM app, similar to the source code.

$$\text{Levenshtein Similarity} = 1 - \frac{\text{Levenshtein Distance}}{\text{Maximum String Length}} \quad (1)$$

where the **Maximum String Length** is the length of the longer string. This allowed us to compare app pairs with different text lengths. We focused on app pairs where the Levenshtein similarity scored between 0.95 and 1.0, excluding exact matches (similarity = 1). For example, with an `instructions` field of 500 characters, fewer than 25 modifications (5% of the total length) would flag potential plagiarism, and for fields of 1000 characters, fewer than 50 changes would trigger detection. This threshold effectively captured minor variations while avoiding false positives due to insignificant changes. To ensure the rigor of our analysis, we excluded comparisons where the `instructions` field was shorter than 50 characters, filtering out trivial entries such as single words or short phrases. This ensured that our analysis focused on substantial content replication. Focusing on high-similarity pairs enabled us to detect apps with minimal textual differences, suggesting potential attempts to clone content while avoiding exact duplication.

2) BERT-based semantic similarity calculation

To detect more nuanced instances of app cloning, where the wording might vary while the underlying meaning remains consistent, we employed a BERT-based model [23] to compute semantic similarity. Unlike character-based methods, this model **utilizes contextual embeddings to capture the semantic closeness between two pieces of text**, allowing for the detection of deeper, more subtle forms of copying. The BERT model maps each input text into a high-dimensional vector space, where semantically similar texts have closer vector representations. Given two texts, t_1 and t_2, their semantic similarity score is calculated using the cosine similarity of their vector embeddings:

$$\text{Cosine Similarity}(t_1, t_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|} \quad (2)$$

where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the embedding vectors generated by the BERT model for texts t_1 and t_2, respectively. The cosine similarity score ranges from 0 to 1, with higher values indicating greater semantic similarity.

We set a threshold of 0.95 for semantic similarity, meaning that if two texts scored above this value, they were flagged as having a strong semantic resemblance. This high threshold ensures precision, reducing the likelihood of false positives, while still capturing relevant instances of duplication. Similar to the Levenshtein distance method, we excluded LLM apps where the `instructions` fields were shorter than 50 characters. Additionally, due to model limitations, we excluded

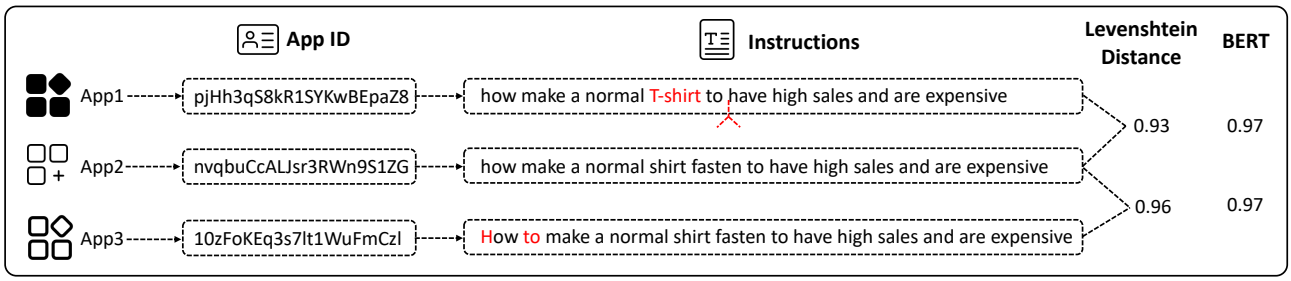


Fig. 5: A real-world example highlights the differences between Levenshtein and BERT-based semantic similarity methods. Although all three apps convey the same core meaning, a typographical error with the term "fasten" in App2 and App3 causes the Levenshtein method to detect similarity only between these two, missing the similarity between App1 and App2.

LLM apps with `instructions` fields that exceeded 512 bytes in length. Unlike the Levenshtein method, however, we did not exclude app pairs with identical `instructions` fields, as these cases still provided valuable insights into semantic consistency.

When the text's meaning remained consistent but the wording varied, the BERT-based approach was more effective than character-based methods. For example, consider three apps, as shown in Figure 5. The Levenshtein method misses the similarity between App1 and App2 due to minor text variations, while the BERT model effectively captures the semantic consistency across all three apps, demonstrating its advantage in detecting deeper similarities.

V. Measuring Impersonation Apps

In this section, we use LLMappCrazy to analyze impersonation apps in LLM app stores, focusing on squatting and cloning. Our investigation is guided by the following RQs:

RQ1 To what extent are squatting apps present? Do they primarily target popular apps? We aim to analyze the prevalence of squatting apps in LLM app stores and determine whether they target more popular apps.

RQ2 How widespread is cloning apps, as another form of impersonation, in LLM app stores? The low barrier to creating LLM apps has allowed cloning apps in LLM app stores to emerge. Our goal is to investigate the prevalence of these apps and understand their potential impact on users and the ecosystem.

RQ3 How many cases of potential cross-platform plagiarism exist? What are the situations in different stores? This RQ aims to understand how app duplication across platforms impacts the uniqueness and integrity of LLM apps, and whether certain stores are more vulnerable to this issue than others.

A. RQ1: Distribution of Squatting LLM Apps.

In response to RQ1, we explore the prevalence and characteristics of app squatting among LLM apps. Our experiments rely on data from GPTs APP [15], the largest third-party GPT store, which provides rankings for the **top 1000 LLM apps**. This platform is essential for our analysis as it offers a ranking system not available in the official GPT Store [27], making it a representative source. To refine the results and minimize false positives, we applied a filtering process. Apps signed by the same developer but with slight name variations, such as platform-specific versions, were excluded. For example, different releases of an "Image Generator" app by the same developer across platforms were not considered squatting. Additionally, apps with common, non-branded names, like "Image Generator", were filtered out unless their package names followed predefined squatting patterns.

Once the data was extracted, we systematically compared it against the GPT dataset to identify instances of name duplication. This comparison revealed that 7,119 apps shared their names with those found in the top 1000 apps, suggesting a widespread occurrence of potential app squatting behavior. Notably, the most frequently duplicated app name was "Prompt Engineer" [4], which appeared 214 times across different records and was ranked 137th, indicating its significant popularity and the possible intent to capitalize on its recognition. Table II below provides an overview of the five apps with the highest number of duplicate names, offering insights into the scale of this phenomenon and the types of apps most often targeted.

TABLE II: Top 5 apps with the most duplicate names.

App Name	Author Name	# Duplicate Name Apps
Prompt Engineer	aitoolreport.com	214
Translator	Caleb Ye	154
Research Assistant	Liseli akayombokwa	129
Resume Builder	masterinterview.ai	127
Logo Creator	None	116

TABLE III: Top 5 apps with the most squatting app names.

App Name	Author Name	# Squatting Name Apps
AI Homework Helper	solvely.ai	132
GPT Store Finder	EmbedAI	126
Study+ Homework Helper	smartprompt.xyz	122
Essay writing assistant	Corine Gorczany	109
Python	Nicholas Barker	106

To further examine the prevalence of app squatting, we utilized our tool LLMappCrazy, to generate various name variations for the top 1000 apps, incorporating common

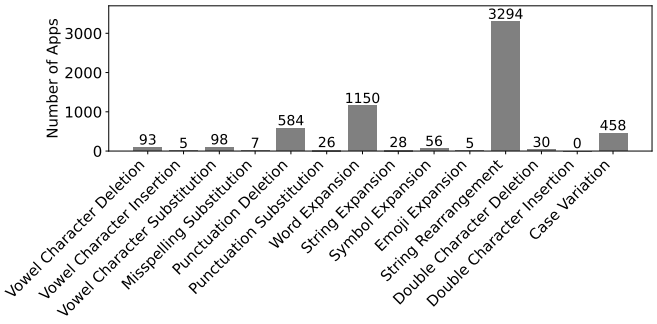


Fig. 6: The distribution of squatting apps across models.

squatting tactics such as case changes, character substitutions, misspellings, and expansions. Using these generated variants, we identified a total of 5,187 apps within the dataset that matched the modified names, highlighting the extensive use of squatting tactics. Table III lists the top 5 apps with the most number of squatting apps. Figure 6 shows the distribution of 5,834 squatting apps across 14 models. The top three patterns, **string rearrangement** (3,294), **word expansion** (1,150), and **punctuation deletion** (584), were newly introduced or modified for LLM apps, proving their effectiveness. Less common patterns like **case variation** (458 apps) highlight additional attack strategies, offering insights for improving detection.

To explore whether squatting apps specifically target more popular LLM apps, we analyzed the distribution of duplicate and squatting apps across different ranks. As shown in Figure 7, higher-ranked apps (closer to the top of the y-axis) have more duplicate and squatting instances, indicated by the denser clustering in the upper region.

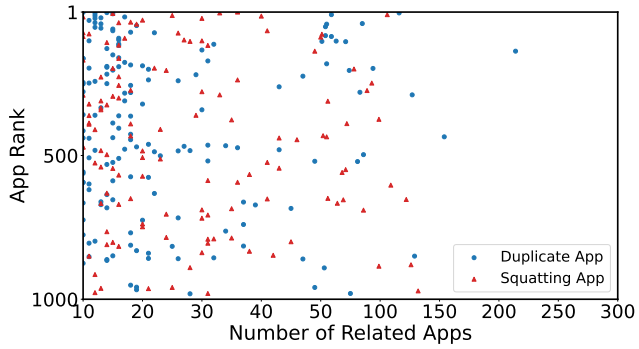


Fig. 7: Distribution of fake and squatting apps by app rank.

As LLM app stores target the general public, most app names are common and familiar, with few uniquely distinctive ones. However, squatting still occurs among these app names. For instance, "logogpts.cn" created an app named "LOGO", and another app, "LOGO+", by "Rodolfo Arce", shares an identical description, suggesting potential squatting. This similarity strongly suggests a potential case of squatting. Nonetheless, we manually filtered out apps with very common names from the top 1000 apps. After filtering, we retained 654 apps and identified a total of 2871 squatting apps.

Answer to RQ1. We found that the top 1,000 LLM apps were associated with 5,834 squatting apps, with more popular apps being more frequently targeted. This could be due to their higher visibility and user demand. The most common method for generating squatting names in LLM app stores involves slight variations of the original app names, i.e. string Rearrangement, word Expansion,

B. RQ2: Prevalence of Cloning LLM Apps.

To address RQ2, we examined app cloning among 785,129 LLM apps from six platforms, focusing on two key fields: the `description` highlights the app's features, while the `instructions` serves as source code. We performed pairwise comparisons of these fields to identify identical or highly similar content, suggesting possible cloning. Our experiments covered both **exact matches** and **semantic similarities**, shedding light on the extent and nature of app cloning within the LLM app ecosystem.

1) Exact match for identical content

We first used exact string matching to detect LLM apps with identical `instructions` or descriptions, effectively identifying direct duplicates where the text was copied verbatim, potentially misleading users into believing these apps are unique. Our analysis revealed significant app cloning across various LLM platforms, with 1,058 apps sharing identical `instructions` and 8,765 apps having identical descriptions. The GPT platform had the highest number of cloned descriptions (7,570 apps), while FlowGPT exhibited the most cloned `instructions` (784 apps). Additionally, 209 apps had both identical `instructions` and descriptions, with intra-platform plagiarism particularly common on platforms like FlowGPT and Poe. Table IV (Columns 3-8) provides a detailed breakdown of these results across all platforms.

2) Similarity detection

As detailed in § IV-C, we used two methods: **Levenshtein distance** and **BERT-based semantic similarity**, to detect app cloning where the `instructions` or descriptions were not identical but still highly similar. These approaches allowed us to identify subtle cloning behaviors, where minor textual changes were made to mask duplication.

Levenshtein distance calculation. Well-suited for detecting subtle variations like minor edits or typos, this method helps identify near-duplicate content. Applying a 0.95 similarity threshold to the `instructions` fields of 42,544 apps (after filtering out those with fewer than 50 characters), we identified 557 groups with high similarity, involving 1,637 apps. As shown in Table V, FlowGPT had the most similar apps (1,396), with approximately 3.84% of apps across the six platforms exhibiting near-duplicate `instructions`. These findings suggest widespread duplication and potential plagiarism, particularly on FlowGPT, warranting further investigation.

BERT-based semantic similarity calculation. To further detect potential app cloning, we applied BERT-based semantic matching to the `instructions` fields, focusing on apps with 50 to 512 characters. This analysis covered 12,048 apps, using a similarity threshold of 0.95. We identified 253 groups

TABLE IV: Overview of cloning apps in six LLM app stores.

Store Name	LLM Apps	Identical Instructions		Identical Descriptions		Identical Both[1]	
	# LLM Apps	# LLM Apps	% of Total	# LLM Apps	% of Total	# LLM Apps	% of Total
GPT Store	662,294	36	0.01%	7,570	1.14%	0	0
FlowGPT	34,271	784	2.29%	944	2.75%	121	0.35%
Poe	16,544	185	1.12%	210	1.27%	76	0.46%
Coze	51,912	33	0.06%	0	0	0	0
Cici	13,060	1	0.01%	1	0.01%	0	0
Character.AI	7,048	20	0.28%	40	0.57%	12	0.17%
Total	785,129	1,058	0.13%	8,765	1.12%	209	0.03%

[1] Identical Both: Number of LLM apps with identical descriptions and instructions.

TABLE V: Results of Levenshtein distance method.

Store Name	Total Detections	Detection Results	Percentage
GPT Store	10,358	22	0.21%
FlowGPT	23,906	1,396	5.84%
Poe	5,177	188	3.63%
Coze	1,429	23	1.61%
Cici	0	0	0
Character.AI	1,674	13	0.78%
Total	42,544	1,637	3.84%

of semantically similar apps, involving 2,113 apps. As shown in Table VI, FlowGPT had the highest number of similar apps (1,705). BERT's ability to capture semantic meaning makes it effective for detecting cloning behaviors that go beyond exact text matches, revealing more nuanced forms of content duplication across platforms.

TABLE VI: Results of BERT-based method.

Store Name	Total Detections	Detection Results	Percentage
GPT Store	1,930	92	4.77%
FlowGPT	5,129	1,705	33.24%
Poe	3,092	258	8.34%
Coze	960	8	0.83%
Cici	0	0	0
Character.AI	937	50	5.34%
Total	12,048	2,113	17.54%

Answer to RQ2. Our findings reveal a high prevalence of cloned apps across LLM app stores, with significant content duplication detected on multiple platforms. We identified 557 groups with highly similar `instructions` and 253 groups based on semantic similarity, involving thousands of apps. These clones pose risks by creating confusion over app authenticity, potentially undermining user trust and the integrity of the LLM app ecosystem.

C. RQ3: Cross-platform Analysis

We analyzed app similarities across multiple LLM app stores to understand how duplication affects the uniqueness and integrity of LLM apps and whether certain stores are more vulnerable. By tracking platform information, we identified cross-platform plagiarism through app groups spanning different platforms. In the cloning experiment, we found 13 groups with identical `instructions`, 130 groups with identical descriptions, and 8 groups where both matched across platforms. Using the Levenshtein distance method, we identified 22 groups of suspected plagiarism, while BERT-based semantic matching revealed 40 groups with deep similarities, even when wording was altered. These findings highlight the complexity of cross-platform plagiarism, where cloning often involves subtle modifications preserving core content. The heatmaps in Figure 8a and Figure 8b show that plagiarism is most common among FlowGPT, Poe, and GPT Store, with significant clustering suggesting these platforms are more prone to cloning and squatting compared to others.

Answer to RQ3. Our analysis identified numerous cases of cross-platform plagiarism, with 13 groups sharing identical `instructions`, 130 groups with identical descriptions, and 8 groups matching in both. Additionally, 22 groups showed high similarity via Levenshtein distance, while BERT analysis found 40 groups with deep semantic overlap. FlowGPT, Poe, and GPT Store were particularly affected, suggesting these platforms are more prone to cloning and squatting, raising concerns about the integrity of LLM apps.

VI. Threat and Impact

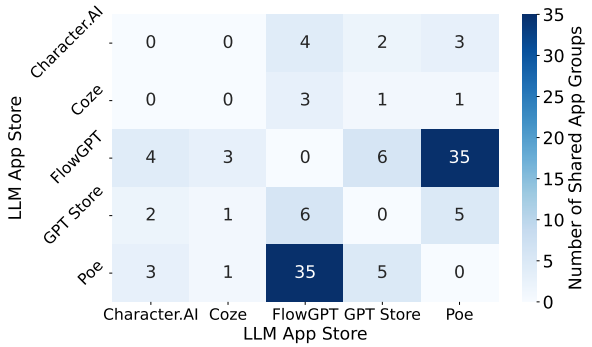
we then examine the threat posed by impersonation apps and their impact on users and the LLM app ecosystem by exploring the following research questions:

RQ4 How many impersonation (squatting and cloning) apps are malicious? Understanding how many of these squatting and cloning apps are malicious will provide insight into the extent of harm they can cause, such as spreading malware or conducting phishing attacks.

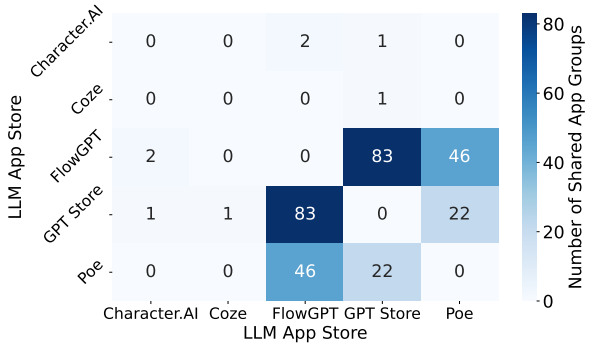
RQ5 What is the impact of these impersonation apps on users and the LLM app ecosystem? This RQ seeks to assess how impersonation apps affect user trust and security, as well as their broader impact on the LLM app ecosystem's integrity.

A. RQ4: Malware Presence

When certain apps exhibit a very high degree of similarity in the fields of `app name`, `description`, and `instructions`, it is clear that these apps are deliberately imitating others, strongly suggesting an intent to impersonate.



(a) Cross-platform detection result of squatting.



(b) Cross-platform detection result of cloning.

Fig. 8: Cross-platform detection result.

To quantify this, we conducted a comprehensive analysis of the squatting and cloning experiment results from RQ1 and RQ2 and identified 227 apps that met the criteria for high similarity. Following this, we aimed to evaluate the potential malicious behavior within squatting and cloning apps. Out of the 3,509 squatting apps and 9,575 cloning apps identified, we selected a representative sample of 347 squatting and 370 cloning apps, using a 95% confidence level and a 5% confidence interval to ensure statistical significance. This sample underwent manual inspection to detect malware, phishing, and ad injection, assessing the risks these impersonation apps might pose to users. Figure 9 illustrates the proportion of malicious apps identified in the sample.

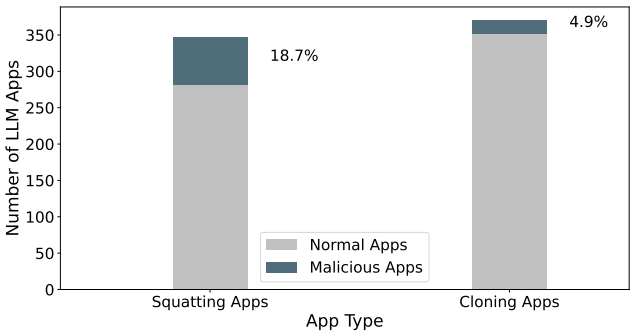


Fig. 9: Result of sampling analysis.

After a thorough manual inspection of 347 selected squatting apps, we found 18.7% violating LLM app usage policies [17]. Of these, 2% provided `instructions` encouraging guideline violations, and 0.3% linked to an unknown website, raising phishing concerns. Alarmingly, 16.4% apps directed users to generate inappropriate content, including sexual, violent, or illegal material. In the 370 cloned apps, 4.9% were non-compliant with LLM policies. Among them, 0.5% encouraged violations, and 3.5% promoted inappropriate content. Notably, 0.8% exhibited fraudulent behavior, claiming to operate "fully automated with a high win rate" to lure users with false promises. As shown in Figure 10, the malicious behaviors detected in our study fall into three categories: **policy violations**, **inappropriate content**, and **disinformation**, with inappropriate content being the most prevalent. Apps promoting illegal content, misleading users, or encouraging policy violations pose serious risks to user safety and data security, undermining trust in LLM platforms and the app ecosystem. If left unchecked, these apps could normalize unethical practices and attract more malicious actors. Our findings highlight the urgent need for stricter regulations and robust monitoring in LLM app stores to ensure user protection and maintain ethical standards, fostering a secure and trustworthy environment.

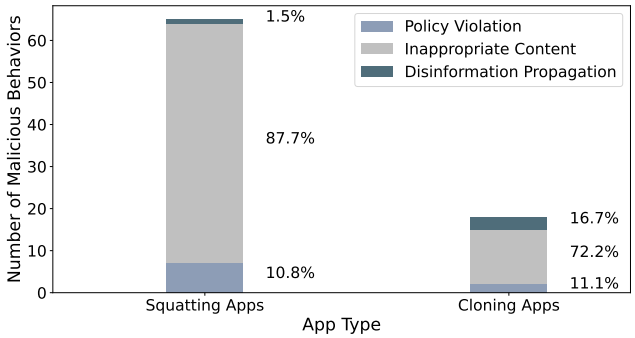


Fig. 10: The distribution of malicious behaviors.

Answer to RQ4. We found that 227 apps exhibited high similarity in `app name`, `description`, and `instructions`, indicating deliberate impersonation. Additionally, among the examined apps, 65 out of 347 squatting apps and 18 out of 370 cloning apps were found to be non-compliant. These apps often provided `instructions` that violated policies, generated inappropriate content, or engaged in fraudulent practices, underscoring significant security risks and the urgent need for stronger regulations to protect users and the ecosystem.

B. RQ5: Impact on Users

Squatting apps in LLM app stores have reached high usage levels, significantly affecting users. Of the 3,509 identified squatting apps, 2,835 had conversation counts between 0 and 1,000, showing a large portion with lower engagement. However, 674 apps exceeded 1,000 conversations, and 50 surpassed 50,000, demonstrating substantial user interaction. In particular, the top squatting app had 12,969,368 conversations, while another app with nearly identical instructions ranked third

with 4,236,464 conversations. These two apps, published by different creators, suggest potential unauthorized replication, posing risks due to high engagement. For cloning apps, of the 9,575 identified, 7,828 had conversation counts between 0 and 1,000, while 1,747 recorded over 1,000, and 726 exceeded 100,000, highlighting significant user interaction. The top cloned app reached 27,527,998 conversations. Figure 11 shows the distribution of conversation counts, with squatting apps peaking broadly at higher counts (around 10^2 to 10^5) and cloning apps peaking sharply at lower counts (around 10^1), indicating that squatting apps generally achieve higher user engagement and visibility, thus posing a greater threat.

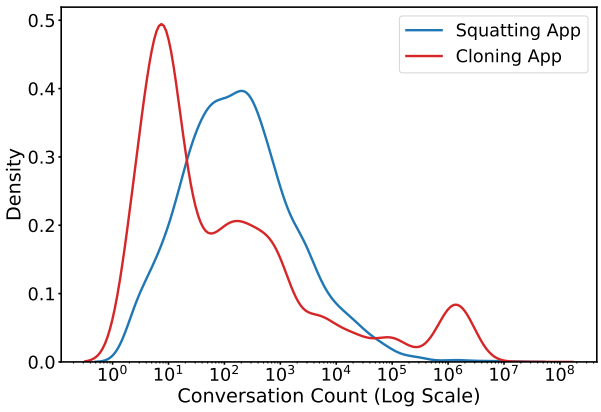


Fig. 11: Kernel density distribution of conversation counts.

High conversation volumes in squatting and cloning apps increase the risk of users unknowingly interacting with unauthorized or low-quality replicas, exposing them to unverified content, potential malicious activities, and privacy issues. These counterfeit apps often lack updates and support, leading to a poorer user experience and overshadowing legitimate apps, complicating access to authentic resources and weakening trust in LLM app stores.

Answer to RQ5. Squatting and cloning apps in LLM stores demonstrate high user engagement, significantly affecting user experience and platform integrity. Of 3,509 squatting apps, 674 had over 1,000 conversations, with the top app reaching 12.9 million interactions. Similarly, 1,747 of 9,575 cloning apps exceeded 1,000 conversations, with the most-used app hitting 27.5 million. Interaction with unauthorized replicas exposes users to security risks and diminishes visibility for legitimate apps, highlighting the need for stronger oversight.

VII. Discussion

A. Mitigation & Implications

We propose strategies to address the challenges of LLM app squatting and cloning, focusing on three key stakeholders:

LLM app store managers. Platforms should enhance their app review processes by incorporating automated and manual checks to detect duplicate or similar apps. Advanced plagiarism detection tools can help identify potential plagiarism during the submission process. Additionally, recommendation algorithms should be improved to prioritize unique, high-quality content and reduce the visibility of cloned apps, ensuring that users encounter a wider variety of original options.

LLM app developers. Developers should take an active role in protecting their apps from squatting and cloning. This includes selecting distinct, non-conflicting app names and regularly monitoring for potential infringements. If unauthorized replicas are found, developers should report these to the platform maintainers to ensure prompt action.

End users. Educating users about the risks of cloned or unauthorized apps is crucial. They should be taught to identify suspicious apps and use tools to verify legitimacy. Developers and platforms can help by offering resources like tutorials and reports to guide users in avoiding squatting attacks and choosing legitimate apps.

B. Threat to Validity

Identical app name detection. Unlike traditional mobile app squatting detection, our approach includes identical app names in LLM app stores, where duplicates are allowed. Squatting attackers tend to use exact names to mimic legitimate apps and deceive users. Including identical names helps detect as many squatting apps as possible. As many developers choose names casually, this can lead to unintentional duplication and false positives. To better distinguish intentional squatting from accidental duplication, we combine squatting and cloning detection based on both name and instruction similarity.

Popular app selection. Our detection of LLM app squatting focuses mainly on the GPT Store, as it is the only platform with app ranking data. This research targets popular apps, which we believe is appropriate since attackers tend to focus on well-known applications. However, future work will examine how to generalize our findings to more LLM apps.

Tool limitation. Although LLMappCrazy is specifically tailored for LLM apps, the generation model may still be incomplete, leaving room for other complex squatting methods. To address this, we designed the squatting generation models in LLMappCrazy as an easily extensible tool, allowing new patterns to be added seamlessly. In cloning detection models, due to input length limitations, we only analyzed instructions of a specified length, potentially missing cloning in apps with longer instructions. However, our results still provide initial evidence of cloning in the LLM app ecosystem, and we plan to improve our detection methods in the future.

Cross-platform deduplication Different authors may use different names across platforms, and in our cross-platform plagiarism analysis, we can only accurately identify cases where the author names are identical. This limitation highlights the need for additional verification processes to distinguish between legitimate cross-platform distribution and unauthorized replication by third parties.

VIII. Conclusion

In this study, we conducted the first large-scale analysis of LLM app squatting and cloning using our tool, LLMappCrazy. Through the detection of 14 squatting generation techniques

and leveraging both Levenshtein distance and BERT-based semantic analysis, we identified over 5,000 squatting apps from variations of top app names. Across six major platforms, we found 3,509 squatting apps and 9,575 cloning cases. Our sampling revealed that 18.7% of the squatting apps and 4.9% of the cloning apps exhibited malicious behavior, highlighting significant risks to user security and the integrity of LLM app stores. These findings underscore the need for stronger oversight and protective measures in the LLM app ecosystem.

Data availability

The artifact is publicly accessible at https://anonymous.4open.science/r/LLM_app_squatting_and_cloning-6000/.

References

[1] "Dnstwist: domain name permutation engine." https://github.com/elceef/dnstwist/, 2018.

[2] "Urlcrazy," .https://www.morningstarsecurity.com/research/urlcrazy, 2018.

[3] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis, "Seven months' worth of mistakes: A longitudinal study of typosquatting abuse." in *NDSS*, 2015.

[4] aitoolreport.com, "Prompt-engineer," https://gptsapp.io/gpts/prompt-engineer-pro-ai/1ql83vbvr, 2024.

[5] S. Antebi, N. Azulay, E. Habler, B. Ganon, A. Shabtai, and Y. Elovici, "Gpt in sheep's clothing: The risk of customized gpts," *arXiv preprint arXiv:2401.09075*, 2024.

[6] Character.Ai, "Character.ai," https://character.ai/, 2024.

[7] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 175–186.

[8] S. Chen, L. Fan, C. Chen, M. Xue, Y. Liu, and L. Xu, "Gui-squatting attack: Automated generation of android phishing apps," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2551–2568, 2021.

[9] Cici, "Cici," https://www.ciciai.com/chat/, 2024.

[10] Claude, "Claude," https://claude.ai/login, 2024.

[11] Coze, "Coze," https://www.coze.com/, 2024.

[12] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on android markets," in *Computer Security–ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings 17*. Springer, 2012, pp. 37–54.

[13] FlowGPT, "Flowgpt," https://flowgpt.com/, 2024.

[14] Gemini, "Gemini," https://gemini.google.com/, 2024.

[15] gptsapp.io, "Gpts app," https://gptsapp.io/, 2024.

[16] D. J. Griffiths, "Detecting cybersquatting-based business email compromise," Ph.D. dissertation, University of Southampton, 2022.

[17] X. Hou, Y. Zhao, and H. Wang, "On the (in) security of llm app stores," *arXiv preprint arXiv:2407.08422*, 2024.

[18] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Gptzoo: A large-scale dataset of gpts for the research community," *arXiv preprint arXiv:2405.15630*, 2024. [Online]. Available: https://arxiv.org/abs/2405.15630v1

[19] Y. Hu, H. Wang, R. He, L. Li, G. Tyson, I. Castro, Y. Guo, L. Wu, and G. Xu, "Mobile app squatting," in *Proceedings of the Web Conference 2020*. International World Wide Web Conferences Steering Committee, 2020, pp. 1234–1245.

[20] Y. Hu, G. Xu, B. Zhang, K. Lai, G. Xu, and M. Zhang, "Robust app clone detection based on similarity of ui structure," *IEEE Access*, vol. 8, pp. 77 142–77 155, 2020.

[21] B. Hui, H. Yuan, N. Gong, P. Burlina, and Y. Cao, "Pleak: Prompt leaking attacks against large language model applications," *arXiv preprint arXiv:2405.06823*, 2024.

[22] M. Khajezade, F. H. Fard, and M. S. Shehata, "Evaluating few-shot and contrastive learning methods for code clone detection," *Empirical Software Engineering*, vol. 29, no. 6, p. 163, 2024.

[23] M. V. Koroteev, "Bert: A review of applications in natural language processing and understanding," 2021. [Online]. Available: https://arxiv.org/abs/2103.11943

[24] Z. Lin, J. Cui, X. Liao, and X. Wang, "Malla: Demystifying real-world large language model integrated malicious services," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[25] F. Lyu, Y. Lin, J. Yang, and J. Zhou, "Suidroid: An efficient hardening-resilient approach to android app clone detection," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 511–518.

[26] H. Niu, T. Yang, and S. Niu, "Clone analysis and detection in android applications," in *2016 3rd International Conference on Systems and Informatics (ICSAI)*. IEEE, 2016, pp. 520–525.

[27] OpenAI, "Chatgpt," https://openai.com/chatgpt/, 2024.

[28] ——, "Gpt store," https://chat.openai.com/gpts, 2024.

[29] Poe, "Poe," https://poe.com/, 2023.

[30] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Information and Software Technology*, vol. 55, no. 7, pp. 1165–1199, 2013.

[31] N. Shazeer and D. Freitas, Daniel, "Character.ai," https://character.ai/, 2024.

[32] J. Spaulding, D. Nyang, and A. Mohaisen, "Understanding the effectiveness of typosquatting techniques," in *Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2017, pp. 1–8.

[33] D. Su, Y. Zhao, X. Hou, S. Wang, and H. Wang, "Gpt store mining and analysis," *arXiv preprint arXiv:2405.10210*, 2024.

[34] J. Szurdi and N. Christin, "Email typosquatting," in *Proceedings of the 2017 internet measurement conference*, 2017, pp. 419–431.

[35] G. Tao, S. Cheng, Z. Zhang, J. Zhu, G. Shen, and X. Zhang, "Opening a pandora's box: Things you should know in the era of custom gpts," *arXiv preprint arXiv:2401.00905*, 2023.

[36] M. Taylor, "Defending against typosquatting attacks in programming language-based package repositories," Master's thesis, University of Kansas, 2020.

[37] M. Taylor, R. K. Vaidya, D. Davidson, L. De Carli, and V. Rastogi, "Spellbound: Defending against package typosquatting," *arXiv preprint arXiv:2003.03471*, 2020.

[38] D.-L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, "Typosquatting and combosquatting attacks on the python ecosystem," in *2020 ieee european symposium on security and privacy workshops (euros&pw)*. IEEE, 2020, pp. 509–514.

[39] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Wukong: A scalable and accurate two-phase approach to android app clone detection," in *Proceedings of the 2015 international symposium on software testing and analysis*, 2015, pp. 71–82.

[40] Wikipedia, "Cybersquatting," https://en.wikipedia.org/wiki/Cybersquatting, 2024.

[41] C. Yan, R. Ren, M. H. Meng, L. Wan, T. Y. Ooi, and G. Bai, "Exploring chatgpt app ecosystem: Distribution, deployment and security," *arXiv preprint arXiv:2408.14357*, 2024.

[42] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[43] W. Zhang, S. Guo, H. Zhang, Y. Sui, Y. Xue, and Y. Xu, "Challenging machine learning-based clone detectors via semantic-preserving code transformations," *IEEE Transactions on Software Engineering*, vol. 49, no. 5, pp. 3052–3070, 2023.

[44] Z. Zhang, L. Zhang, X. Yuan, A. Zhang, M. Xu, and F. Qian, "A first look at gpt apps: Landscape and vulnerability," *arXiv preprint arXiv:2402.15105*, 2024.

[45] Y. Zhao, X. Hou, S. Wang, and H. Wang, "Llm app store analysis: A vision and roadmap," *arXiv preprint arXiv:2404.12737*, 2024.