Careful About What App Promotion Ads Recommend! Detecting and Explaining Malware Promotion via App Promotion Graph

Shang Ma[†*], Chaoran Chen[†*], Shao Yang[‡], Shifu Hou[†],
Toby Jia-Jun Li[†], Xusheng Xiao[§]✉, Tao Xie[¶], Yanfang Ye[†]✉
[†]University of Notre Dame. Email: {sma5, cchen25, shou, jli26, yye7}@nd.edu
[‡]Case Western Reserve University. Email: sxy599@case.edu
[§]Arizona State University. Email: xusheng.xiao@asu.edu
[¶]Peking University. Email: taoxie@pku.edu.cn

***Abstract*—In Android apps, their developers frequently place app promotion ads, namely advertisements to promote other apps. Unfortunately, the inadequate vetting of ad content allows malicious developers to exploit app promotion ads as a new distribution channel for malware.**

To help detect malware distributed via app promotion ads, in this paper, we propose a novel approach, named ADGPE, that synergistically integrates app user interface (UI) exploration with graph learning to automatically collect app promotion ads, detect malware promoted by these ads, and explain the promotion mechanisms employed by the detected malware.

Our evaluation on $18,627$ app promotion ads demonstrates the substantial risks in the app promotion ecosystem. The probability for encountering malware when downloading from app promotion ads is hundreds of times higher than from the Google Play. Popular ad networks such as Google AdMob, Unity Ads, and Applovin are exploited by malicious developers to spread a variety of malware: aggressive adware, rogue security software, trojan, and fleeceware. Our UI exploration technique can find 24% more app promotion ads within the same time compared to the state-of-the-art techniques. We also demonstrate our technique's usage in investigating underground economy by collecting app promotion ads in the wild. Leveraging the found app promotion relations, our malware detection model achieves a 5.17% gain in F1 score, improving the F1 score of state-of-art techniques from 90.14% to 95.31%. Our malware detection model also detects 28 apps that were initially labeled as benign apps by VirusTotal but labeled by it as malware/potentially unwanted apps (PUAs) six months later. Our path inference model unveils two malware promotion mechanisms: custom-made ad-based promotion via hardcoded ads and ad library-based promotion via interactions with ad servers (e.g., AdMob and Applovin). These findings uncover the critical security risks of app promotion ads and demonstrate the effectiveness of ADGPE in combining dynamic program analysis with graph learning to study the app promotion ad-based malware distribution.

I. INTRODUCTION

Advertisements, in short as ads, are widely used in mobile apps. Research [1] indicates that over 57% of apps in Google Play contain ad libraries. Additionally, two-thirds of popular apps contain these ad libraries [2]. Ads are effective and widely adopted practices to increase user bases and app installs, and are also a major revenue source for developers who place ads in their apps to promote other apps (denoted as app promotion ads in this paper) [3], [4]. These ads play a significant role in helping users discover new apps. For example, a survey [5] from Google Research and Ipsos shows that one-third of users discover new apps via ads in other apps. Unfortunately, the inadequate vetting of the ad content allows malicious developers to exploit app promotion ads as a new distribution channel for malware.

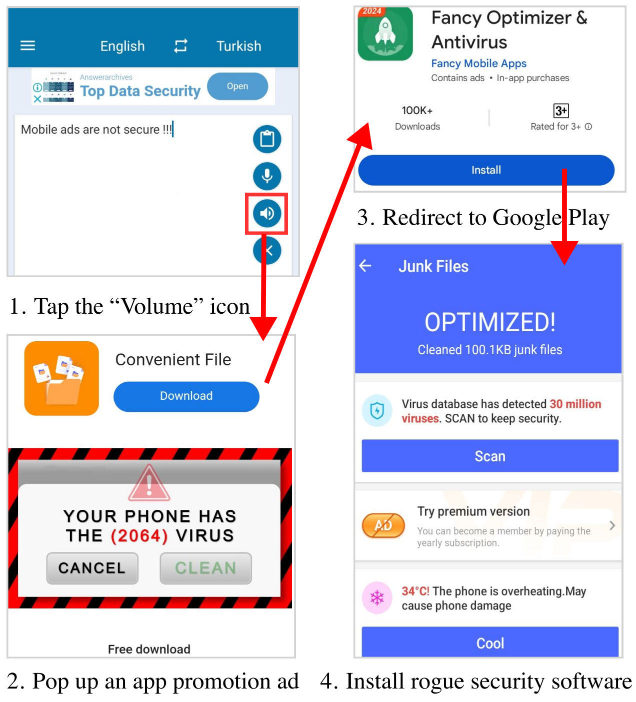

English
Turkish
Top Data Security
Open
Mobile ads are not secure !!!
1. Tap the "Volume" icon
Convenient File
Download
YOUR PHONE HAS THE (2064) VIRUS
CANCEL
CLEAN
Free download
2. Pop up an app promotion ad
Fancy Optimizer & Antivirus
Fancy Mobile Apps
Contains ads • In-app purchases
100K+
Downloads
Rated for 3+
Install
3. Redirect to Google Play
Junk Files
OPTIMIZED!
Cleaned 100.1KB junk files
Virus database has detected 30 million viruses. SCAN to keep security.
Scan
Try premium version
You can become a member by paying the yearly subscription.
34°C! The phone is overheating.May cause phone damage
Cool
4. Install rogue security software

Fig. 1: Malware distribution via an app promotion ad

App users are prone to malware promotion ads for two main reasons. First, these ads are usually "hidden" in harmless apps, engaging users with ad contents. For example, the app shown in Fig. 1 offers free translation services between English and

✉ Corresponding authors. * Equal contribution.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230051
www.ndss-symposium.org

Turkish. Nevertheless, any interaction with this app, such as tapping the "Voice" icon, will trigger an app promotion ad. This ad employs persuasive language and visual patterns to encourage users to download rogue security software that convinces users to pay for security services and steals users' sensitive information. Second, the malware promotion ads are usually indistinguishable from other ads, leading users to unknowingly download malware from trusted sources. In Fig. 1, tapping anywhere on the ad redirects users to the official Google Play store to install the advertised app. Moreover, this ad is provided by Google AdMob, which is the most widely used ad library, integrated into over 90% of apps that conduct app promotion campaigns. Similar malware promotion ads are also observed from other popular ad libraries such as Meta Ads and AppLovin.

To help detect malware distributed via app promotion ads, in this paper, we propose a novel approach, ADGPE, that **synergistically integrates app UI exploration with graph learning to automatically collect app promotion ads, detect malware promoted by these ads, and explain the promotion mechanisms employed by the detected malware**[1]. In particular, while static analysis can effectively identify ad libraries used to show ads in apps [6]–[8], our preliminary study finds that applying static analysis to detect app promotion ads is insufficient because (a) the ad content served by ad libraries is determined at runtime and dynamically requested from ad servers, and (b) custom-made ads (ad content provided by the app developer instead of ad libraries) that account for non-trivial portions of ads ($> 20\%$) have diverse implementation mechanisms (Section III-C). Thus, to address these challenges, ADGPE performs automatic user interface (UI) exploration on a large number of apps from Google Play to collect app promotion ads, and leverages the collected ads to build ***an app promotion graph***. In an app promotion graph, the nodes represent apps, and the edges represent the relationships from the apps showing the ads to the apps promoted by the ads. This graph encapsulates the app promotion relations among apps and the app attributes derived from app markets, VirusTotal, and source code. Utilizing the graph, ADGPE then trains a node classification model for malware detection and constructs a path inference model to uncover malware promotion mechanisms.

ADGPE starts by collecting app promotion ads to construct an app promotion graph. We have pinpointed three major challenges of this task. First, app promotion ads are often placed on multiple UI pages within an app. Existing UI exploration techniques aim at achieving high code coverage within a single app by employing complex and heavy strategies, which become inefficient when scaling to a large number of apps [9]–[13]. Second, app promotion ads appear in various formats, each with distinct UI layout characteristics. Current works on identifying mobile ads are limited to a small set of UI widgets thus failing to capture all the app promotion ads [8], [14], [15]. Third, ads' content is generated at runtime and continuously changes over time. Existing UI exploration techniques are optimized on code coverage so they tend to avoid repeated actions and may overlook the changed ad content [16]–[19]. To address these three challenges, ADGPE includes a novel ad-oriented UI exploration technique. First, to efficiently achieve high ad coverage, this technique employs random exploration combined with a depth-first exhaustive search strategy. Second, to accurately identify various ad types, this technique leverages a common design feature: a call-to-action widget redirecting users to the app marketplace. Third, to capture the frequently updated ad content, this ad-oriented UI exploration technique periodically restarts apps and refreshes their ads to capture all available content within a period of time.

The second part of ADGPE is to leverage the constructed app promotion graph for ad-promoted malware detection. Existing Android malware detection models have two shortcomings when detecting ad-promoted malware [20]–[22]. First, they are ineffective as they mainly train learning models using only app attributes (e.g., permissions, app metadata, and API calls) and *overlook the app promotion relations*. Second, these models generally *lack explainability*. To address these shortcomings, ADGPE introduces a novel graph learning model based on pre-trained Graph Neural Network (GNN) [23]–[26]. To effectively detect ad-promoted malware, this model constructs an embedding of the app promotion graph to represent app promotion relations. This embedding is then combined with the app attributes to train a Random Forest classifier [27] for malware detection. To enhance the explainability of our model, ADGPE transforms the app promotion graph into a promotion inference graph (PIG) and builds a path inference model to infer malware promotion mechanisms based on app attributes and app promotion relations. The path inference model also predicts unobserved links in the app promotion graph, complementing UI exploration to build a more comprehensive app promotion graph for better detection and inference.

Our study on 18,627 app promotion ads uncovers significant risks in these ads: a 2.64% chance for users to encounter malware when downloading apps via app promotion ads, which rises to 7.51% when the ads are in potentially unwanted apps (PUAs). This probability is ***hundreds of times higher than the likelihood of downloading a malicious app from the Google Play store***. Furthermore, our case study reveals that ***popular ad libraries such as Google AdMob, Unity Ads, and Applovin lack stringent vetting processes for malware and are exploited by malicious developers to spread a variety of malware: aggressive adware, rogue security software, trojan, and fleeceware***. App developers can also build custom-made ads to distribute malware. Although app markets have strict policies, they alone cannot mitigate all risks without cooperation from ad libraries and developers. Thus, solely analyzing ad libraries is not sufficient.

By incorporating app promotion relations, our malware detection model obtains a 5.17% ***performance gain*** (from 90.14% to 95.31%) compared to using solely traditional features[2]. This indicates that the app promotion graph built

[1]ADGPE is publicly available at https://github.com/AppPromotionAdsResearch/AdGPE.

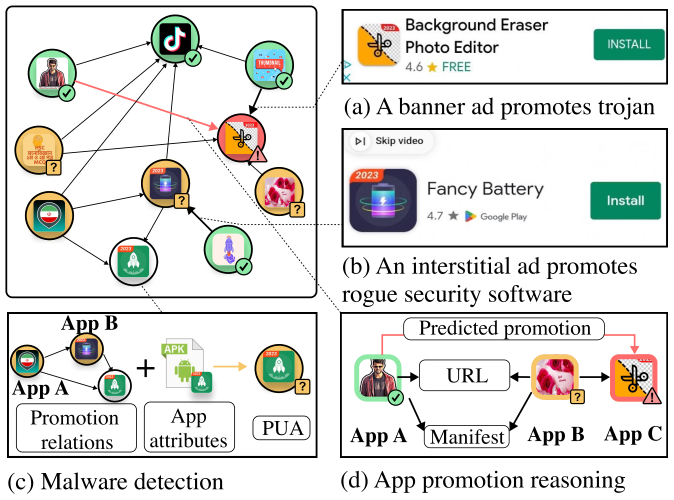
[2]In this paper, the term "malware", when used in the context of malware detection or malware promotion, encompasses both the malware and PUAs.

through UI exploration is an important feature that enhances the effectiveness of graph learning in malware detection. Specifically, it outperforms five commercial security engines by at least 27.4% and two state-of-the-art (SOTA) malware detection approaches [28], [29] by at least 24.5% in F1 score. Additionally, ADGPE achieves a better F1 score than the SOTA approach that employs random walk [30] and is fed with the same features of ADGPE (95.31% v.s. 92.48%), indicating the superiority of ADGPE's GNN model. We evaluate the robustness of ADGPE by mimicking real-world attacks through mutating nodes and links in the app promotion graph. The results demonstrate that ADGPE maintains a high F1 score with node mutations and outperforms previous work with link mutations. Our malware detection model also successfully detects 28 apps that were initially labeled as benign apps by VirusTotal but labeled as malware/PUAs six months later. Furthermore, our ad-oriented UI exploration technique outperforms four established approaches [8], [31]–[33], by finding 24% more ads within 10 hours. Our small-scale case study on collecting app promotion ads in the wild also shows that this UI exploration technique can effectively detect ads directing users outside Google Play and discover malware from non-official app markets, assisting in studying underground economy. Our path inference model reveals two primary malware promotion channels: ad library-based promotion via interactions with ad servers (e.g., AdMob, Applovin) and custom-made ad-based promotion via hardcoded ads (e.g., apps from the same malicious developer) in the app's source code. This path inference model also aids ad-oriented UI exploration by predicting missing app promotion links, some of which require at most 54 clicks to find. These findings demonstrate ***the effectiveness of combining dynamic program analysis with graph learning in studying app promotion ad-based malware distribution.***

II. Background and Threat Model

Mobile Advertisement Ecosystem. The mobile advertising ecosystem includes several key roles: *Ad Networks*, which conduct real-time auctions to determine which ads to display based on factors such as bid price, ad quality, and user context [34], [35]; *Ad Providers*, who aim to promote their apps via ads; and *Developers*, who seek to monetize their apps by hosting ads. Ad providers must publish their apps on an app market and register these apps with an ad network to finance advertisement campaigns [36]. Ad networks, which are incorporated into mobile apps via ad libraries [37], [38], support various formats like banners and rewarded ads to enhance user visibility [39]–[41]. To host ads, developers need to register their apps with an ad network and integrate the network's ad library into their apps [42]. Additionally, to reduce advertising expenses, developers can create custom-made ads within their own apps to promote other products or apps they offer.

Automatic UI Exploration. Automatic UI exploration for Android devices is an automated approach that systematically navigates and interacts with an app's UI to identify potential issues, validate functionality, and ensure a seamless user experience [43], [44]. Recognizing the importance, Google



(a) A banner ad promotes trojan
(b) An interstitial ad promotes rogue security software
(c) Malware detection
(d) App promotion reasoning

Fig. 2: A motivating example

has released the UI exerciser, the Monkey tool [32], which is a command-line tool to randomly generate user events including clicks, touches, and gestures. There also exist research tools that model the explored UI pages and transitions to automatically explore various behaviors of apps [11]–[13], [45]. Apart from these automated UI testing tools, there are also UI testing techniques that enable developers to create customized code to automate the UI tests, such as Appium [46] and Google's UIAutomator [47].

Threat Model. We conduct the study in our controlled lab environments with trusted devices and secure network settings, and our threat model is consistent with the previous work [48], [49]. Under this lab setting, we assume that app promotion ads in our experiments are not subject to external attacks. Specifically, we assume that the ads are generated by the apps themselves instead of from other apps that preempt the ad promotion channel to distribute ads in the other apps. We also assume that the network channels and the ad servers of the apps are not compromised and the ad content is not altered. Although such scenarios may occur, they fall outside the scope of this research. To ensure that the annotation of the app class (i.e., benign app, PUA, and malware) is accurate and our trained model is not biased or polluted, we not only rely on the VirusTotal security report of each app, but also sample and inspect the source files of the reported malware to ensure the existence of malicious behavior in the code. Malware that deliberately evades detection and inspection can be detected by applying more advanced techniques [50]–[54], which is out of the scope of this paper.

III. Motivation of ADGPE

In this section, we introduce the motivation behind our approach, starting with an example that highlights the potential risks associated with app promotion ads. We also use an example to illustrate the benefits of integrating graph learning with UI exploration, demonstrating the enhanced effectiveness of this synergistic approach.

A. Risks in App Promotion Ads

Fig. 2 shows a real-world app promotion graph where apps are connected by app promotions ads (arrows pointing to promoted apps). This graph consists of malware (the icon with a red triangle), PUAs (icons with a yellow rectangle), and benign apps (icons with a green circle). The malware “Background Eraser Photo Editor” and the PUA “Fancy Battery” can both be accessed by users following the app promotion ads seen in Fig. 2(a) and (b), both of which are located in benign apps. The malware “Background Eraser Photo Editor” is a trojan that disguises itself as a photo editor while executing malicious activities such as stealing sensitive information and e-banking frauds. The PUA “Fancy Battery” is a rogue security engine that disguises itself as a cleaner app but executes malicious services automatically to constantly display ads without requiring user interactions.

Beyond leveraging ad libraries, malware is also promoted through custom-made app promotion ads. Fig. 7a in the Appendix shows an example app that pops up a dialog in a benign app and directs users to download a trojan. *Clearly, these examples show that malware/PUAs can exploit app promotion ads in benign apps, which are trusted by most users, to reach more users.*

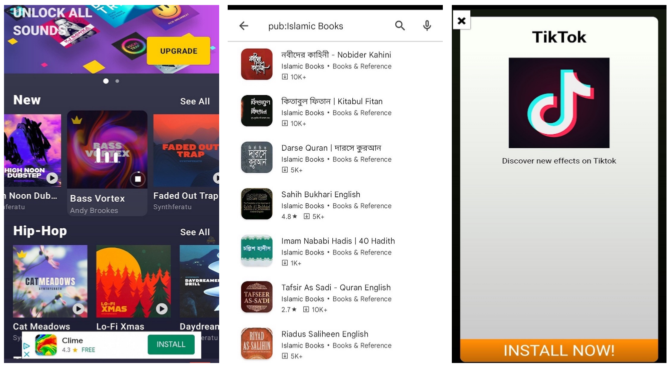
B. Graph Learning with App Promotion

Fig. 2(c) presents an example of using app promotion relations to detect malware. Specifically, the adware “Learn Persian”, which promotes the target app, “Fancy Boost”, also promotes another PUA named “Fancy Optimizer & Antivirus”, both developed by the same developer. Such app promotion relations suggest a high likelihood that the target app belongs to the same group of malicious developers (referring to Section VII-B for more details). The target app “Fancy Boost” indeed exhibits unwanted behaviors similar to the rogue security software “Fancy Optimizer & Antivirus” (as shown in Fig. 1). Nevertheless, this app is a newly developed app with limited information and security vendors such as VirusTotal classify it as benign. This example shows how app promotion relations facilitate graph learning-based malware detection by providing distinctive features.

Fig. 2(d) shows how to use an app promotion graph to predict and reason app promotions. Specifically, the online message sticker app “Vijay Deverakonda Sticker” (referred to as App A) accesses the same ad library URL and shares manifest activity names with the photo collection app “Morning and Night Wishes” (referred to as App B), which is known to promote the malware “Background Eraser Photo Editor” (referred to as App C). This information implies that App A and App B share exploit code and use the same ad library, suggesting a strong likelihood of App A promoting App C. However, real-time UI exploration may not capture every instance of app promotion ads, potentially missing this malware promotion. This example demonstrates that graph learning offers a solution by predicting app promotions, effectively complementing dynamic analysis to address its inherent limitations, such as the incompleteness of app UI exploration.

```java
import com.google.android.gms.ads.AdView;
import com.google.android.gms.ads.InterstitialAd;
// Inherent ads
AdView adView = findViewById(R.id.adView);
adView.loadAd(ConsentSDK.getAdRequest(this));
// Pop-up ads
InterstitialAd popupad = new InterstitialAd(this);
popupad.loadAd(ConsentSDK.getAdRequest(this));
adButton.setOnClickListener(view -> popupad.show());
// Custom-Made ads
customAdButton.setOnClickListener(view -> startActivity(new
      Intent(Intent.ACTION_VIEW, Uri.parse(uri))));
```

Fig. 3: Key code snippets of three ad types



(a) Inherent Ad (b) Custom-made Ad (c) Pop-up Ad

Fig. 4: Examples of app promotion ads

C. Characterizing Challenges for App Promotion Ads

App promotion ads are shown in various formats [39]–[41], including banner, interstitial, and rewarded ads. Moreover, app promotion ads can be implemented by *ad libraries* or *custom-made by developers*, posing challenges for collecting app promotion ads. Therefore, we conduct a preliminary study to investigate the characteristics of app promotions ads and how we can collect them. We have built two datasets as follows:

AndroZoo Dataset. We first construct a dataset of advertiser apps based on AndroZoo [55], a comprehensive collection of APK files that is regularly updated and widely acknowledged in the research community. Our selection criteria from AndroZoo are as follows: 1) the apps must have been released after 2020 to ensure that their advertising practices are current; 2) they must include ad libraries, indicating a likelihood of containing advertisements; 3) they should have a Play Store page to largely guarantee the app’s quality; 4) their download in Play Store should be diversified, as apps with different popularity may have different advertising practices. We filter the dataset with these criteria and then randomly sample 200 apps of the resulting dataset. We run these apps on Android phones and manually explore these apps to record the app promotion ads and the app promoted by those ads.

Rico Dataset. Manually exploring app promotion ads in a large number of apps from AndroZoo requires extensive human labor. To support a comprehensive study, we additionally use Rico dataset [56], which provides screenshots, view hierarchies and human annotations. The screenshots and corresponding view hierarchies directly reveal the GUI information of app promotion ads, such as the attributes of the ad-related widget and the placement of the ad-related

TABLE I: Dataset of the preliminary study

Ad types	Using ad libraries	# Samples		Example
		AndroZoo	Rico	
Inherent ads	Yes	75	167	Fig. 4a
Pop-up ads	Yes	69	22	Fig. 4c
Custom-Made ads	No	45	5	Fig. 4b

widget in the UI. Human annotations (including interaction traces) tell the location of ad-related UI in the whole app and how to reach it. Specifically, we refine this dataset to include only screenshots labeled as "Advertisemen", yielding 1326 screenshots with view hierarchies from 405 apps. We then manually review these screenshots and view hierarchies.

Categorization of Ad Types. We browse the documentation of ad libraries [39]–[41] and also manually decompile the advertiser apps to inspect the code logic triggering app promotion ads. Based on our observations, we categorize the recorded ads into three ad types. Fig.3 illustrates the key code snippets for each ad type, while Fig.4 provides examples of these ad formats.

- *Inherent ads:* These ads are shown using visible views created via ad libraries (Line 1 and Line 4), and are usually shown when apps are started.
- *Pop-up ads:* These ads are created using ad libraries (Line 2 and Line 7) and are usually triggered by user interactions, such as clicks (Line 9).
- *Custom-Made ads:* These ads are developed by app developers and can be of various forms such as buttons, banners, or app walls. The ad content (such as the URI of the advertising site shown in Line 11) can be hardcoded or dynamically requested from developers' own servers.

It can be seen that ad content from both inherent ads and pop-up ads are dynamically requested from the ad library via `getAdRequest` (Lines 5 and 8), and thus cannot be analyzed by static analysis. It can also be challenging for static analysis to analyze custom-made ads due to the highly customized implementations from different developers.

Ad Characteristics for Malware Promotion. Table I summarizes the characteristics of these ad types and their distribution in the two datasets. From the AndroZoo dataset, we can find that while ads from ad libraries (i.e., inherent and pop-up) are the majority, custom-made ads account for 23.81% (45/189) of the ads, indicating that custom ads represent a non-trivial portion of the ads analyzed. Furthermore, we have scanned the promoted apps via VirusTotal, and the results reveal that custom-made ads are much more likely to promote malware than the inherent and pop-up ads: 51.11% (23/45) of the custom-made ads promote malware while only 34.78% (24/69) of pop-up ads and 42.67% (32/75) inherent ads promote malware. *This finding demonstrates the necessity of analyzing custom-made ads rather than focusing solely on ads from ad libraries.*

Motivation for UI Exploration. For ad-library ads (i.e., inherent and pop-up ads), we aim to explore whether the same ad libraries exhibit different ad promotion behaviors across apps (i.e., different apps with the same ad libraries promoting different apps). We characterize the diversity of the promoted apps based on three aspects: functionality (app category in the app market), maliciousness (whether the app is malware), and popularity (whether the download number exceeds 1 million). We first conduct a Pearson's chi-square test [57] to explore the association between the use of specific ad libraries and the diversity of promoted apps, Specifically, we employ a lightweight static analysis [6] to detect the ad libraries in an app. We find that the most frequently used ad libraries in our dataset are AdMob [36], AppLovin [41], and Facebook Ads [40]. For each advertiser app utilizing these libraries, we randomly sampled 10 (or all, if fewer than 10) apps that they promoted. The statistics reveal no significant association between the ad libraries used in the advertiser apps and the app category, maliciousness, or popularity of the promoted apps. In contrast, we discovered a significant association between the categories of advertiser apps and those of the promoted apps with a p-value of $6.02e^{-05}$. These findings lead us to conclude that the promotion behaviors of advertiser apps are influenced more by the individual advertiser apps than by the specific ad libraries used, aligning with findings from existing studies [58]. This motivates us to develop UI exploration techniques for individual apps, regardless of the ad libraries used.

IV. App Promotion Graph Construction

In this section, we present the ad-oriented UI exploration technique in detail and describe how to construct the app promotion graph.

A. Ad-Oriented UI Exploration

Based on the preliminary study, AdGPE employs a dynamic UI exploration to collect app promotion ads for building an app promotion graph rather than relying on static analysis on ad libraries. In particular, based on the characteristics of app promotion ads, we have observed three unique challenges:

- *Mobile GUI Navigation*: Identifying ad-related UI is challenging due to ads often appearing after navigating through multiple UI pages, especially when analyzing a large number of apps [11]–[13]. Thus, the technique needs to effectively and efficiently navigate to the ad-related UI.
- *App Promotion Ad Detection*: App promotion ads appear in various formats, each with distinct UI layout characteristics. Thus, we need to design a precise mechanism in detecting app promotion ads when they show up in the UIs.
- *Dynamic Ads Capture*: Ads' content is generated at runtime and continuously changes over time. For example, relaunching the same app after a certain period may display a different banner in the same location, promoting a different app. Thus, UI exploration needs to repetitively visit the same UI pages to collect more ads.

We next describe how our ad-oriented UI exploration technique addresses these challenges.

Mobile GUI Navigation. While previous studies focused only on ads displayed on the main page UI [8], our preliminary study on the AndroZoo dataset reveals that over 28% of ads appear in UIs other than the main page. User interaction data from the Rico dataset also indicates that, on average, a user must engage in 8 interactions to reach a UI containing ads .

To scale up the exploration to a large number of apps while still maintaining high coverage of ads, we adopt a random exploration method combined with a depth-first exhaustive search strategy. This strategy is shown in previous studies to be more effective in finding deeper exploration paths that are more likely to identify more ads [11], [32], [45]. Other more complex strategies such as program analysis techniques [9], [10] or UI model refinement [11]–[13] require heavier weight computation during UI exploration and cannot easily scale to a large number of apps. Specifically, our technique initiates a recursive search of all UIs, starting from the main UI upon app launch. It interacts with all widgets (e.g., touch, scroll, and select). If interaction with a widget leads to navigation outside the app (excluding Google Play), our technique returns to the previous activity to explore another unexplored widget. If navigation occurs within the same app, it continues interacting with widgets in the current activity. This process continues until all widgets in the app's UIs have been interacted with or until a preset timeout is reached. Following the existing practices, we also pre-register several accounts and write scripts to bypass login [31], since many apps cannot navigate to the main page without passing the login.

App Promotion Ads Detection. A direct approach to detecting app promotion ads is to search for ad-related keywords in the UI hierarchies of explored UIs. However, our analysis of the AndroZoo dataset reveals that the naming practice of ad-related widgets actually varies across different apps and developers, lacking a consistent pattern. Additionally, we find that app promotion ads are displayed using more than 15 UI widgets types (e.g., the most common ones are View, TextView, Button). Thus, existing UI exploration techniques [8], [14], [15] that focus on a small set of UI widgets such as `WebView` and `ImageView` fail to detect all the app promotion ads.

To address this challenge, we observe that all ads have the same UI design goal: encouraging users to tap on the ads to download promoted apps. For example, the usage of the green "install" button of the banner ad shown in Fig. 4a, and the orange "INSTALL NOW!" button for the intersititial ad shown in Fig. 4c. Building on these observations, we construct a list of ad-specific keywords containing those encouraging words we encountered most frequently during the empirical study. Based on this list, our technique prioritizes the exploration of the widgets whose attributes (e.g., (`text`, `resource-id`, and `class`)) contain any of these keywords.

After interacting with a widget, users are generally redirected to Google Play or third-party websites for app downloads. Note that Google Play does not provide direct APK downloads, and some promoted apps may be unavailable in the market due to local policy violations. To address this, we collect the redirection link and extract the package name of the promoted app from the link. For third-party websites, we manually open the link and record the package name of the promoted app.

Dynamic Ads Capturing. We observe that when an app is restarted, its app promotion ads may change (i.e., they promote a different app). To investigate how the ads are dynamically altered, we keep restarting an app to refresh its ads. We observe that, within a specific time period, the total number of unique ads obtained from an app remains constant [59]. This suggests that ad libraries infrequently update their recommendation lists, instead maintaining them for a period of time.

TABLE II: Attributes from different sources

Source	Name	Brief Description
App Market	App Name	The name of the app
	Developer Name	The name of the developer
	Reviews	Count of user reviews
	Downloads	Count of downloads
	Star	Average star rating
	Description	Developer-provided app description
	Rating	Age-based content rating (e.g., Teen, 18+)
	Category	App category (e.g., Social, Tools)
VirusTotal	Flags	Count of malware flags
	Report	Results of security engine analyses
	URL	URLs from the "Interesting String" field
Code	Manifest	Content of the AndroidManifest.xml file
	API Calls	API calls extracted from smali code
	Signature	Hash of the app's certificate signature

To address this challenge, we run our exploration technique on each app within the predefined time limit (5 minutes in our experiment), and iteratively restart the app to repeat the detection. We record the promoted apps from the ads until no new promoted apps are identified after a pre-defined number of iterations (3 in our experiment) or the maximum number of iterations is reached (20 in our experiment). We adopt this setting because it can effectively capture most of the ads recommended by the ad library within a time range.

B. Graph Construction

To construct an app promotion graph, we apply our ad-oriented UI exploration technique on the seed dataset detailed in Section VI-B. Specifically, for each promoted app, we install the app based on the recorded package name from the AndroZoo dataset. We then run the exploration technique on the app to collect more app promotion ads. This process repeats until no new apps are found in the app promotion ads. With the collected app promotion ads, we are able to construct an app promotion graph to map out the promotion relationships within the ecosystem.

To facilitate a more in-depth analysis, we enrich the graph by collecting additional data about each app as attributes of the graph's nodes. This additional data is sourced from three distinct origins, as detailed in Table II.

App Market Attributes. To investigate the relationship between app promotion and various factors such as organization, popularity, and functionality, we crawl the Google Play Store pages of each app and extract information such as the name of the app, the number of total downloads, and the app category.

VirusTotal Attributes. To construct the groundtruth of malware and PUAs in the app promotion graph, we query VirusTotal for the security flags of each app and obtain the corresponding vendor reports. VirusTotal requires the APK file or its corresponding hash to perform an analysis. Hence, we crawl the relevant SHAs from AndroZoo. Notably, multiple SHAs may be associated with the same package name. To ensure a conservative analysis, we select the five most recent

SHAs (if available) and choose the SHA with the highest number of VirusTotal flags for our analysis.

Code Attributes. The ad libraries have unique code-level characteristics for their special system and network behaviors [8], [60]. Hence, we obtain the API calls from the smali code of each app. Furthermore, we download the APK file of the promoted app from AndroZoo and decompile it using Androguard [61]. We next extract component names from the decompiled manifest file. Component names reveal reused code, such as that from the same ad library or developer. We also extract the signature, which conveys the app's organizational information.

V. Malware Detection and App Promotion Reasoning

As shown in Fig. 5, we introduce a pre-trained graph embedding to represent each node by aggregating its neighborhood information, i.e., app promotion relations. The pre-trained node embedding is then used to perform two downstream tasks: *malware detection* and *app promotion reasoning*.

A. Pre-trained Graph Embedding

For a given node a in the app promotion graph, we categorize the apps' attributes into three groups based on Table II: textual, numerical, and categorical attributes.

- *Textual attributes*: Textual attributes describe the developer, the functionality, and the purposes of an app, including the attributes of "App Name", "Developer Name", and "Description". To encode these attributes, we remove stopwords, apply stemming, and then employ the Term Frequency-Inverse Document Frequency (TF-IDF) [62] technique to transform them into feature vectors, denoted as v_t^a.
- *Numerical attributes*: Numerical attributes represent an app's popularity, including the attributes of "Reviews", "Downloads", and "Star". We standardize them to form the feature vector, denoted as v_n^a.
- *Categorical attributes*: Categorical attributes contain the attribute of "Manifest", including the activities, providers, services, receivers, permissions, and the attributes of "Rating", and "Signature". We use one-hot encoding to represent them, denoted as v_c^a.

We also encode the attribute of "API Calls" as v_a^a by leveraging the technique of MaMaDroid [28]. The feature vectors of these four app attributes are then concatenated as the app attribute embedding $v_{app}^a = CONCAT(v_t^a, v_n^a, v_c^a, v_a^a)$.

We employ GraphSAGE [24] to aggregate app promotion relations. Formally, for each node a, its one-hop neighbors are denoted by $N(a)$. We apply a mean aggregator, which takes the element-wise mean of the app attribute embedding $v_{app}^u, u \in N(a)$, to generate the aggregated node embedding v_{agg}^n, which is defined as follows:

$$v_{agg}^{N(a)} = ReLU(W_1 \cdot MEAN(v_{app}^u, u \in N(n_i))) \quad (1)$$

$$v_{agg}^a = ReLU(W_2 \cdot CONCAT(v_{app}^a, v_{agg}^N(a)) \quad (2)$$

where W_1 and W_2 are the learnable parameters of GraphSAGE. We compute the v_{agg}^a for each node as the pre-trained graph embedding that represents the semantics not only from the app attributes but also the app promotion relations, which enables the downstream tasks on the app promotion graphs.

B. Malware Detection

Malware detection refers to detecting whether a promoted app is PUA/malware. Conventional solutions in the industry (e.g., VirusTotal) rely on signature analysis. Consequently, they often fail when the apps in question are freshly developed or recently updated, which is common for apps promoted by ads. Moreover, existing academic Android malware detection studies [21], [22], [28], [29] do not consider app promotion relations, making them less effective in detecting malware through app promotions.

To address this challenge, we leverage both app attributes and app promotion relations from our app promotion graph to develop a comprehensive malware detection model. Specifically, we concatenate the app attribute embedding v_{app} and the pre-trained node embedding v_{agg} to form a unified feature vector for each app. This feature vector captures both the app attributes and the promotion relations. We further concatenate the united feature vectors for both the advertiser app and the promoted app as the final vector to enhance the app promotion information. Due to the nature of the app promotion ecosystem, our input data are imbalanced, and benign apps take a majority of them. To mitigate this issue, we used Random Forest [27] as our final classifier. As an ensemble method, Random Forest combines the results of multiple decision trees, thereby reducing noise and overfitting caused by the imbalanced data.

C. App Promotion Reasoning

The app promotion graph contains rich information on app attributes and app promotion relations. To reveal how such information contributes to promoting specific apps, especially malware, AdGPE transforms the app promotion graph into a promotion inference graph (Pig) and establishes a path inference model to infer malware promotion mechanisms. Particularly, this path inference model works together with our malware detection model to enhance AdGPE's capability in analyzing app promotion ads, and we summarize the benefits of our path inference model as the following two aspects:

- *Enhancing Explainability in Malware Detection*: existing AI techniques for malware detection suffer from a lack of explainability [63]. Our path inference model addresses this gap by inferring malware promotion mechanisms (e.g., ad libraries, shared developers). The observed recurring mechanism increases the confidence of our malware detection.
- *Predicting Missing App Promotion Links*: Without access to the complete list on the ad server, our UI exploration technique is limited in capturing all app promotion links. Moreover, even when such links are identified, the process can be time-consuming. Our path inference model steps in to predict these missing links, streamlining the construction of the app promotion graph. Inferring unseen malware promotion paths also aids in identifying malware with similar promotion mechanisms.

Pig Construction. We build the Pig by extracting the app attributes as entities and adding the relations as denoted in

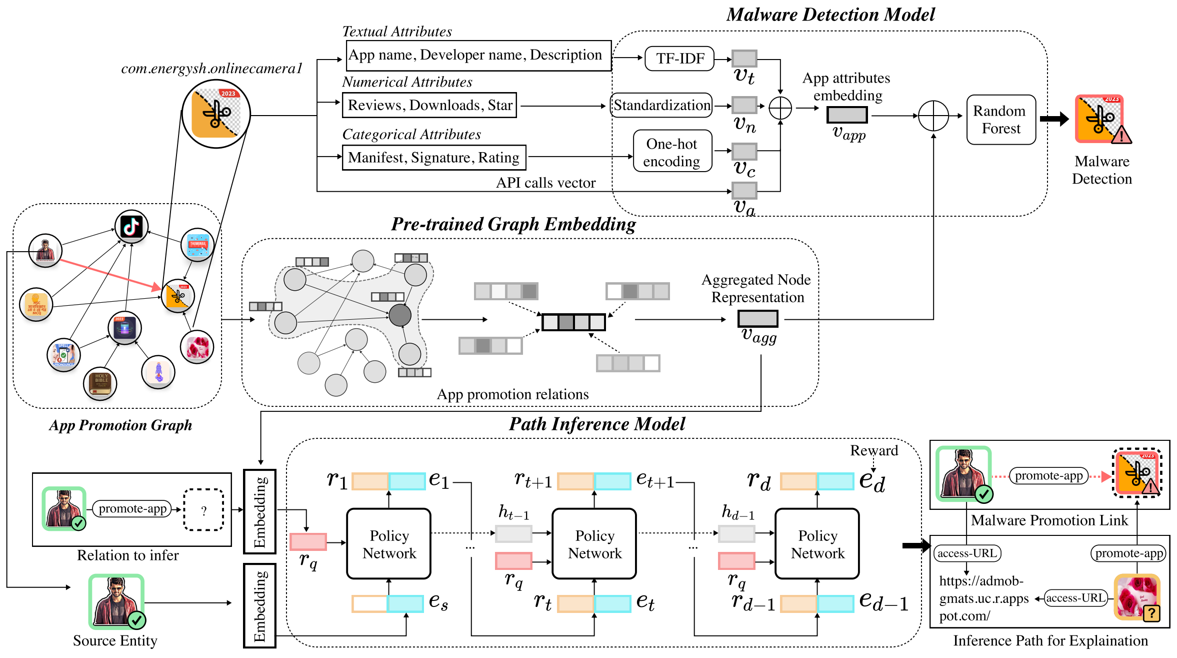


Fig. 5: Overview of malware detection and promotion reasoning

TABLE III: Promotion Inference Graph (PIG) relations

Relation	Definition
R1: promote-app	An app promotes another via app promotion ads
R2: has-sig	An app has a digital signature, verifying its authenticity and integrity
R3: has-manifest	An app has a manifest file containing activities, providers, receivers, services, and permissions
R4: access-URL	An app has access to a URL
R5: isin-category	An app belongs to an app category in Google Play
R6: VT-flag	A security engine flags an app as malware
R7: create-app	A developer creates an app
R8: develop-category	A developer creates an app belonging to a category
R9: use-URL	A developer creates an app which accesses a URL

Table III. Formally, we define PIG $= (E, R)$, where E is the set of entities and R is the set of relations.

Path Inference We train a path inference model to infer the malware promotion mechanisms and predict missing links over PIG, as shown in Fig. 5. Following the model structure of Multi-Hop [64], our model employs a reinforcement learning algorithm [65]. This is designed to infer the destination entity $e_d \in E$ based on the input source entity $e_s \in E$ and the relation $r_q \in R$ we desire to infer. At each time point t, the model decides an entity $e_t \in E$ and the relation $r_t \in R$ that leads to e_t. Eventually, the model outputs an inference path $(e_s, r_1, e_1, ..., r_t, e_t, ..., r_d, e_d)$ to connect e_s and e_d, where the relation between e_s and e_d aligns with r_q.

We use word2vec [66] to convert the indices of the source entities and relations into a vector form. We then augment the entity representation by concatenating the vectors of app entities with our pre-trained graph embedding v_{agg}. For other entities and relations without v_{agg}, we append vectors of zeros to maintain a uniform representation.

During the training time, we employ a policy network [65] to output entities and relations for each time point t. Specifically, the input of the policy network consists of the pair of relation and entity $(e_t,\ r_t)$ and the hidden state h_{t-1}, which represents the search history $(e_s, r_1, e_1, ..., r_{t-1}, e_{t-1})$. Given the desired relation r_q, the policy network then samples the pair of relation and entity $(e_{t+1},\ r_{t+1})$ for the next time point and outputs the search history h_t. This process repeats until it reaches the destination entity e_d. To ensure accurate path-finding, the policy network gets a reward of 1 for observed triplets $(e_s,\ r_q,\ e_d)$ in the PIG. Otherwise, it gets an approximated soft reward [64]. Soft rewards account for the app promotion graph's dynamic and incomplete nature, enabling the model to identify unseen paths in training, enabling the model to predict missing links. Eventually, the traversed relations and entities constitute the inference path $(e_s, r_1, e_1, ..., r_t, e_t, ..., r_d, e_d)$, explaining the potential reason for app promotion.

During the inference time, we input pairs of source entities and relations. The path inference model encodes them through the policy network, then performs a breadth-first search to decode and find destination entities and generates the inference paths. If the output entities contain malware, we review the inference path and identify the promotion mechanisms. To diversify destination entities, we apply a post-processing technique [64] that computes a list of unique destination entities and assigns the maximum score among all paths leading to each unique entity. The final outputs are the top-ranked unique destination entities with the inference path.

VI. Evaluation Setup

A. Research Questions

We seek to evaluate the effectiveness of ADGPE in constructing the app promotion graph and detecting the ad-promoted malware. Specifically, we aim to answer the following research questions:

- **RQ1:** How effectively can AdGPE detect malware promoted by ads?
- **RQ2:** How apps, especially malware, are promoted in the constructed app promotion graph?
- **RQ3:** How the apps with app promotion ads found by AdGPE evolve across time? Among these apps, are there any new apps that have not been scanned by VirustTotal?
- **RQ4:** How effectively can AdGPE build the app promotion graph?
- **RQ5:** How effectively can AdGPE's path inference model predict promotion links? Can these predicted links be used to reason about how apps are promoted through ads?

We next present our dataset and implementation, and then describe our evaluation findings in detail.

B. Dataset

Seed Dataset. We collect apps from AndroZoo as our study subjects since AndroZoo is well-maintained and regularly updated with various versions of apps. Within our affordable effort, we target the apps released from January 1st, 2018 to February 3rd, 2023. Following the practice of the existing works [8], [67], we curate the dataset for three app classes based on the malware flags from VirusTotal: (1) *Malware*, which is flagged by at least 10 engines; (2) *PUAs*, which are flagged by 1-9 engines; and (3) *Benign Apps*, which are not flagged by any engine. By distinguishing between "PUA" and "malware", we can analyze how potentially harmful apps are exploited to promote those with actual malicious behaviors, gaining more understanding of the app promotion ecosystem. In total, we construct a seed dataset consisting of $36,000$ apps, with $12,000$ for each app class.

App Promotion Graph Dataset. Table IV shows a summary of our dataset. Out of the $36,000$ seed apps, we successfully executed $15,344$ apps on our devices. As some apps were released as early as 2018, they are not properly upgraded for the recent Android versions and cannot be executed. Most of the other apps failed the executions as they crashed during the launch. In the executed apps, our UI exploration technique identifies $2,420$ apps that have app promotion ads and further collects $3,859$ additional apps promoted by them. Among these, there are 271 apps that not only have app promotion ads but are also promoted by other apps.

In total, our constructed app promotion graph consists of $6,008$ nodes/apps, including 464 malware, $1,042$ PUAs, and $3,961$ benign apps. Note that the total number of apps ($6,008$) is not a sum of the number of apps ($5,467$) in the malware, PUA, and benign classes. This discrepancy is due to some promoted apps (541) not being archived in AndroZoo at the time of our analysis, which we will further elaborate in Section VII-C. From these apps, our technique collects $18,627$ instances of app promotion ads. In particular, 22.2% of the ads are linked to malware (520) and PUAs ($3,616$), while the remaining ads are tied to benign apps ($13,054$). Additionally, our technique records screenshots, timestamps, and interaction traces during app exploration to confirm and evaluate the collected app promotion ads.

TABLE IV: Dataset summary

# Seed Apps	$36,000$	# Apps Executed	$15,344$
# Apps Having App Promotion Ads	$2,420$	# Apps Being Promoted	$3,859$
# Nodes of App Promotion Graph	$6,008$	# Links of App Promotion Graph	$18,627$

Malware Detection Dataset. Our malware detection model is trained and tested on the app promotion graph with $5,467$ nodes, i.e., the number of nodes that have VirusTotal labels, and $18,627$ links, using stratified 10-fold cross-validation. The path inference model, based on a graph with $127,100$ entities and $565,739$ relations, is split in an 80:10:10 ratio for training, validation, and testing.

C. Implementation

We implement the ad-oriented UI exploration technique upon DroidBot [31], which has been well-maintained up to November 2023. DroidBot is compatible with all Android API versions and offers an interface for implementing customized exploration strategies. Since ad libraries employ emulator detection mechanisms to prevent fraudulent ad traffic, showing only test ads on emulators, we conduct our app exploration on real devices. Specifically, we use six Samsung Galaxy A13 smartphones running Android 11 for our research, which extends over a period of more than three months. We use mitmproxy to record the redirection links after interacting with UI widgets [68]. We leverage networkx [69] to build the app promotion graph.

To construct the pre-trained graph embedding, we use two GraphSAGE [24] convolution layers with 128-dimension embedding. The dropout rate for each layer is 0.5. We apply the Adam optimizer [70] with learning rate of 0.01 and weight decay of 0.0005. We follow the model architecture in Multi-Hop [64] to implement the path inference model. Each edge is regarded as bidirectional. We only keep the top-256 neighbors for each entity with the highest PageRank [71] scores to prevent GPU memory overflow. The embedding size of all entities and relations is 200. Xavier initialization [72] is used for initializing all neural network layers within the model. The SOTA embedding model, DistMult [73], is employed to attain the soft reward. We apply a three-layer LSTM to encode hidden states, each with a dimension of 200. We apply the Adam optimizer with a learning rate of 0.001 and a mini-batch size of 32. The dropout rate for the neural layers is set at 0.1. The decoding breadth-first search size is 128.

VII. Results

A. RQ1: Malware Detection

Comparison to Baselines. We compare AdGPE's performance against the five security engines of VirusTotal with the best F1 scores. Additionally, we compare AdGPE with three state-of-the-art (SOTA) malware detection approaches: MaMaDroid [28], DroidEvolver [29] and ANDRUSPEX [30]. For a fair comparison, we re-implement these approaches using their open-source code and employ Random Forest classifiers, aligning with their original configurations.

TABLE V: Comparison of malware detection baselines and ablation study. The −promotion denotes AdGPE trained without app promotion relations. The →DGI, →GRACE, and →MVGRL denote AdGPE with the embedding method, GraphSage, replaced with DGI, GRACE, and MVGRL, respectively

	Approaches	Accuracy	Precision	Recall	F1 score
Baselines	Symantec	96.99	81.66	69.01	74.80
	Lionic	96.72	74.64	74.64	74.64
	McAfee	95.99	69.56	67.60	68.57
	Avira	94.26	53.57	84.50	65.57
	K7GW	93.63	50.41	85.91	63.54
	DroidEvolver [29]	$75.48_{\pm 7.12}$	$72.92_{\pm 7.96}$	$70.93_{\pm 11.39}$	$71.21_{\pm 6.96}$
	MaMaDroid [28]	$79.38_{\pm 7.33}$	$75.48_{\pm 6.32}$	$78.41_{\pm 9.54}$	$76.58_{\pm 6.14}$
	ANDRUSPEX [30]	$95.15_{\pm 1.24}$	$95.32_{\pm 1.14}$	$88.79_{\pm 3.19}$	$92.48_{\pm 3.19}$
Ablation Study	− promotion	$96.29_{\pm 1.07}$	$95.27_{\pm 3.68}$	$86.01_{\pm 7.23}$	$90.14_{\pm 7.23}$
	→DGI [74]	$97.47_{\pm 0.61}$	$99.10_{\pm 2.19}$	$91.43_{\pm 6.82}$	$94.96_{\pm 6.82}$
	→GRACE [75]	$97.45_{\pm 0.66}$	$99.82_{\pm 0.55}$	$91.43_{\pm 6.64}$	$95.30_{\pm 6.64}$
	→MVGRL [76]	$97.38_{\pm 0.65}$	$98.57_{\pm 2.48}$	$90.90_{\pm 7.27}$	$94.40_{\pm 7.27}$
	AdGPE	**97.74**$_{\pm 0.62}$	$99.44_{\pm 1.67}$	**91.78**$_{\pm 7.02}$	**95.31**$_{\pm 7.02}$

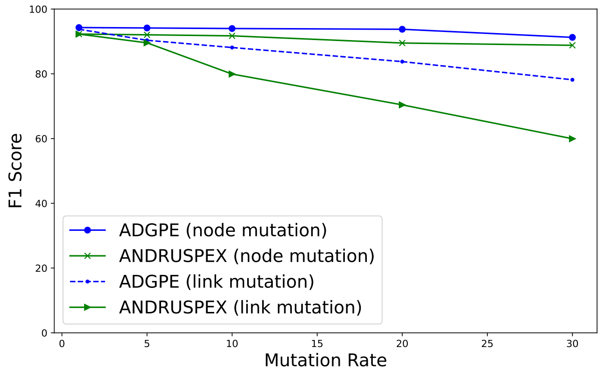


Fig. 6: Robustness comparison for AdGPE and ANDRUSPEX

As illustrated in Table V, AdGPE outperforms all the compared baselines, achieving a 97.74% accuracy, a 99.44% precision, a 91.78% recall, and a 95.31% F1 score. In contrast, though the five security engines achieve high accuracy, all of the baselines suffer from low precision, recall, and F1 score values, suggesting a poor performance in malware detection. The superior performance of AdGPE can be attributed to "App Market" attributes, which are among the most important features of its random forest classifier. For example, some malware is excluded from the Google Play store or exhibits distinctive features such as low star ratings. Overall, these findings demonstrate the superiority of AdGPE over commercial security products and SOTA malware detection approaches.

Ablation Study. We conduct an ablation study to measure the importance of using app promotion relations and compare the effectiveness of different embedding methods for the promotion relations, i.e., DGI [74], GRACE [77], MVGRL [75]. As shown in Table V, using app promotion relations helps detect extra malware samples, as it significantly increases the recall of AdGPE from .01% to 91.78%, and F1 score from 90.14% to 95.31%. Furthermore, while all of them significantly exceed the performance of AdGPE trained without app promotion relations, GraphSage, adopted by our approach, outperforms other embedding methods, *These results demonstrate that app promotion relation is an indispensable feature in detecting ad-promoted malware.*

TABLE VI: Statistical analysis of app promotions by hop distance and app class. Dst/Src represents the destination/source of an app promotion

		# App Promotions			Promotion Probability (%)		
	Src / Dst	Benign	PUA	Malware	Benign	PUA	Malware
Hop=1	Benign	8, 206	3, 052	1, 796	72.53	64.57	69.45
	PUA	2, 997	1, 320	736	26.49	27.92	28.46
	Malware	111	355	54	0.98	7.51	2.09
Hop=2	Benign	14, 735	10, 046	2, 918	73.08	70.82	70.64
	PUA	5, 248	3, 656	1, 133	26.03	25.77	27.43
	Malware	181	484	80	0.90	3.41	1.9
Hop=3	Benign	20, 686	15, 089	4, 250	73.41	70.93	71.55
	PUA	7, 276	5, 685	1, 601	25.82	26.72	26.95
	Malware	216	500	89	0.77	2.35	1.50
Hop=4	Benign	26, 407	18, 384	5, 501	74.19	71.50	73.35
	PUA	8, 915	6, 809	1, 903	25.05	26.48	25.37
	Malware	272	518	96	0.76	2.01	1.28
Hop=5	Benign	30, 731	20, 497	6, 282	74.25	71.75	73.35
	PUA	10, 353	7, 539	2, 185	25.01	26.39	25.51
	Malware	306	532	98	0.74	1.86	1.14

Robustness Analysis. To evade detection, attackers can manipulate the node attributes (e.g., by changing app descriptions of malware) or links (e.g., by letting malware promote benign apps). We therefore evaluate the robustness of AdGPE by mutating nodes and links. Specifically, we apply Gaussian noise to node attributes and perform random link swapping based on the out-degree of each node at rates of 1%, 5%, 10%, 20%, and 30%. As shown in Fig. 6, for node mutations, AdGPE maintains high F1 scores, with a slight drop from 94.29% at 1% to 91.26% at 30%. In contrast, link mutations significantly affect performance, decreasing F1 scores from 93.76% at 1% to 78.16% at 30%. This degradation is more profound in ANDRUSPEX [30], whose F1 score drops from 92.26% at 1% to 59.95% at 30%, demonstrating that AdGPE outperforms ANDRUSPEX in adversarial settings. To defend against such attacks, we can also integrate adversarial training mechanisms by incorporating synthesized adversarial examples in the training data to mitigate the impacts of these attacks.

Case Study on Detected Malware Besides identifying malware promoted via ad libraries, AdGPE additionally identifies malware promoted via custom-made ads, which are overlooked by prior studies [8], [15], [78]. For example, AdGPE identifies a magnet searcher app, available on Google Play [79], as malware. Noteworthy, this app itself does not exhibit any malicious behavior. However, clicking on any search result within the app redirects users to a third-party website where they are prompted to download a malicious downloader [80]. This suggests that the magnet searcher app, which seems harmless on the surface, is actually a malware dropper. This strategy allows it to bypass traditional malware detection models. By using the app promotion graph, AdGPE is able to detect these hidden threats effectively.

B. *RQ2: Malware Promotion*

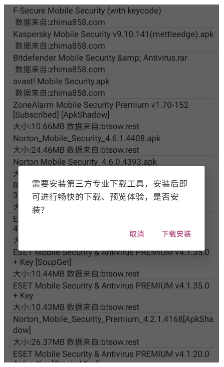
Overall Statistics. To understand how different app classes are promoted in ads, we conduct a statistical analysis of the constructed app promotion graph and quantify the likelihood of an app class promoted by another app class within k hops in the app promotion graph by computing the corresponding promotion probability. For example, the promotion probability of a PUA ($\mathbb{P}$) directly promoting malware ($\mathbb{M}$) at Hop $= 1$ is computed as $\mathrm{P}(\mathbb{P} \rightarrow \mathbb{M}) = \frac{|\mathbb{P} \rightarrow \mathbb{M}|}{|\mathbb{P} \rightarrow \mathbb{M}| + |\mathbb{P} \rightarrow \mathbb{P}| + |\mathbb{P} \rightarrow \mathbb{B}|}$, where $|\mathbb{P} \rightarrow \mathbb{M}|$ represents the number of ads in PUAs that promote malware, $|\mathbb{P} \rightarrow \mathbb{P}|$ represents the number of ads in PUAs that promote PUAs, and $|\mathbb{P} \rightarrow \mathbb{B}|$ represents the number of ads in PUAs that promote benign apps. This yields a promotion probability of $7.51\% = \frac{355}{3052+1320+355}$. Note that we restrict our analysis to at most 5 hops, as extending beyond this limit shows marginal changes in discovering ads promoting new apps.

Malware Promotion. As evidenced by the $Hop = 1$ section of Table VI, merely $\mathrm{P}(\mathbb{B} \rightarrow \mathbb{M}) = 0.98\%$ of ads within benign apps directly promote malware. In contrast, $\mathrm{P}(\mathbb{B} \rightarrow \mathbb{P}) = 26.49\%$ benign apps promote PUAs, which have a substantially larger probability, $\mathrm{P}(\mathbb{P} \rightarrow \mathbb{M}) = 7.51\%$, to further promote malware. This reveals a covert malware promotion route in the app promotion graph. Moreover, our dataset reveals that PUAs either directly ($Hop = 1$) or indirectly ($Hop > 1$) promote the vast majority of malware: 90.46% $(484/535)$ within two hops and 99.25% $(532/535)$ within five hops. *These findings clearly show that engaging with PUAs through app promotions significantly increases the risk of malware installation.*

To better illustrate the risk of malware promotion in ads, we estimate the probability for users to encounter malware in the Google Play market, and compare it to the probability of encountering malware from app promotion ads. Using a conservative calculation, we find that merely 0.002% of all apps $(11,014$ out of $5,151,555)$ released on the Google Play market from January 1, 2018, to February 3, 2023, are malware. Considering that Google Play downloads are predominantly from popular apps [81], the actual probability of encountering malware there is even lower than 0.002%. In contrast, the probability of users encountering a malware promoted by app promotion ads is significantly higher: 7.51% from PUAs, 2.09% from malware, and 0.98% from benign apps. While these probabilities are not very high, due to the huge user base of Google Play [36], [82], a handful of malware, such as trojan, can easily harm millions of users. *As the probability of encountering malware through ads exceeds the Google Play market rates by over* 100 *times, our findings demonstrate that app promotions pose a significant security risk to the users, and more regulations should be applied to vet the apps being promoted in ads.*

Case Study. We further conduct a case study on what are the unwanted behaviors of the promoted PUAs and malware, as well as the strategies employed in their promotions.

- *Trojan:* More than half (53%) of the PUAs/malware are labeled as trojans by VirusTotal. Interestingly, many are labeled as trojans primarily due to the presence of code obfuscation techniques, such as the use of the "jiagu" library, rather than any malicious behavior identified based on our manual analysis. However, our investigation did uncover a subset of genuine trojans (93 in total), all from the developer "lomol language" or "lomol translator." These

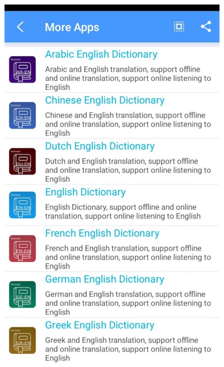

F-Secure Mobile Security (with keycode)
数据来自:zhima858.com
Kaspersky Mobile Security v9.10.141(mettleedge).apk
数据来自:zhima858.com
Bitdefender Mobile Security &amp; Antivirus.rar
数据来自:zhima858.com
avast! Mobile Security.apk
数据来自:zhima858.com
ZoneAlarm Mobile Security Premium v1.70-152 [Subscribed] [ApkShadow]
大小:10.66MB 数据来自:btsow.rest
Norton_Mobile_Security_4.6.1.4408.apk
大小:24.46MB 数据来自:btsow.rest
Norton Mobile Security_4.6.0.4393.apk

需要安装第三方专业下载工具，安装后即可进行畅快的下载、预览体验，是否安装?

取消 下载安装

ESET Mobile Security & Antivirus PREMIUM v4.1.35.0 + Key [SoupGet]
大小:10.44MB 数据来自:btsow.rest
ESET Mobile Security & Antivirus PREMIUM v4.1.35.0 + Key
大小:10.43MB 数据来自:btsow.rest
Norton_Mobile_Security_Premium_4.2.1.4168[ApkShadow]
大小:26.37MB 数据来自:btsow.rest
ESET Mobile Security & Antivirus PREMIUM v4.1.20.0

(a) Trojan Dropper


More Apps

Arabic English Dictionary
Arabic and English translation, support offline and online translation, support online listening to English

Chinese English Dictionary
Chinese and English translation, support offline and online translation, support online listening to English

Dutch English Dictionary
Dutch and English translation, support offline and online translation, support online listening to English

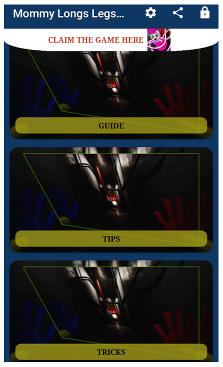
English Dictionary
English Dictionary, support offline and online translation, support online listening to English

French English Dictionary
French and English translation, support offline and online translation, support online listening to English

German English Dictionary
German and English translation, support offline and online translation, support online listening to English

Greek English Dictionary
Greek and English translation, support offline and online translation, support online listening to English

(b) Trojan



(c) Aggressive Adware

Fig. 7: Case study on malware promotion

apps typically masquerade as dictionaries or language translators. For instance, one app disguises itself as a Chinese dictionary and employs code obfuscation techniques that hinder static analysis [83]. By executing the app, we observe that the app scans network information (e.g., reading files under the `/sys/class/net/directory`), and system information (e.g., executing the shell command `ls /`), and try to gain root access by executing `su`.
Promotion Strategy: These "lomol" apps adopt a unique promotion strategy by developing custom-made app promotion ads and using their own ad library (please refer to Fig. 7b for an example) instead of employing the common ad libraries.
- *Aggressive Adware*: Approximately one-third (32%) of PUAs/malware within the app promotion graph are labeled as adware by VirusTotal, primarily due to their aggressive advertising behaviors. Intriguingly, we observe that adware providing similar functionality such as wallpaper downloading and barcode scanning often shares common package names and UIs. For instance, we identify a cluster of 59 PUAs/malware with package names incorporating the title of a popular game followed by terms like "guide," "trick," or "hint" (see Fig. 7c). Additionally, in another group of 13 PUAs/malware, each adware provides downloads of wallpapers related to well-known movies or anime (e.g., Venom, Sonicex).
Promotion Strategy: The main activity names of these apps all contain the string "AppsGeyser", a platform that advertises itself as a rapid app creation tool [84]. This finding suggests the mass-production strategy employed by ad monetizers, leveraging streamlined app creation processes and distributing through app promotion ads to target specific user demographics, such as children and enthusiasts of games and cartoons.
- *Rogue Security Software*: There are five PUAs with "fancyclean" included in their package names, all developed by "Fancy Mobile Apps". These apps pretend to be security engines or phone cleaners but force users to watch 30-second video ads upon performing almost every operation (see Fig. 1).

To verify whether these apps provide the promised functionality, we deploy one of the PUAs, namely "Fancy Security & Antivirus", on a testing phone that we deliberately install viruses, and run the app to scan the phone. We find that the app does not actually scan the phone and simply overlooks the installed viruses. Note that McAfee reported a similar group of "HiddenAds" malware in 2022 [85]. Interestingly, the "fancyclean" apps in our dataset not only exhibit similar behaviors but also have similar package names, app names, and icons to the malware in the report.

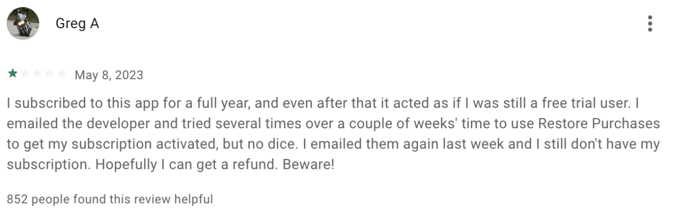
Promotion Strategy: We find that the "Fancy Mobile Apps" developer employs a centralized promotion strategy. The developers leverage their most popular flagship app, "Fancy Battery: Cleaner, Secure" with a Facebook profile to attract traffic and promote other apps [86]. Although it has been removed from Google Play, cached data reveals that this flagship app receives over 13 million downloads.

C. RQ3: Temporal Analysis

We conduct a temporal analysis to understand how app promotion ads evolve over time. Initially, $1,334$ apps were identified to promote either PUA or malware in Feb 2023. Note that the other part of this paper relies on data and findings from this February experiment. We reran the ad-oriented UI exploration technique in August 2023, updating our dataset based on the AndroZoo database, which is refreshed daily. Overall, out of the initial $1,334$ apps, 734 apps still have app promotion ads, while the total number of promoted apps has dropped from $3,212$ to $1,228$. The amount of ads promoting benign apps, PUAs, and malware has decreased by roughly 30%, 45%, and 61%, respectively. This is mainly because the developers stopped maintaining the app promotion ads, often due to the release of newer versions or the apps being removed from Google Play. For instance, the Chinese dictionary trojan, mentioned in RQ2, was found to promote 107 unique apps before its update. However, in August, the February version of the app displayed no app promotion ads while the latest version, released in August displayed such ads.

Zero-Day Apps. Additionally, we discover 190 apps uploaded to Google Play in February with no VirusTotal labels (i.e., this field is set to null) profiled by the AndroZoo dataset, acquiring label values by August. Consequently, the security status of these apps was uncertain in February, leading us to categorize them as zero-day apps. Within this group, 20 apps have gained over one million downloads, and five have even more than 10 million. Moreover, 64 apps, constituting over one-third of the zero-day apps, have either fewer than $1,000$ downloads or have already been removed from the marketplace. Among the zero-day apps, we notice an uptick in chatbot apps leveraging GPT technology: 18 are identified in our dataset, and five have crossed the one million downloads. Despite their popularity, user reviews reveal significant concerns about these chatbot apps, particularly regarding their premium services, which include issues like high costs, convoluted unsubscription procedures, and subpar customer service (see Fig. 8). Some are even labeled as scams or fleeceware by VirusTotal.

Late-Detection Malware. Furthermore, we apply ADGPE's malware detection model on the new dataset generated by this temporal analysis. ADGPE successfully identifies 28 late-detection malware that were labeled as benign apps by


Greg A

May 8, 2023

I subscribed to this app for a full year, and even after that it acted as if I was still a free trial user. I emailed the developer and tried several times over a couple of weeks' time to use Restore Purchases to get my subscription activated, but no dice. I emailed them again last week and I still don't have my subscription. Hopefully I can get a refund. Beware!

852 people found this review helpful

Fig. 8: A user review for "Ask AI - Chat with Chatbot" complaining about the subscription issue

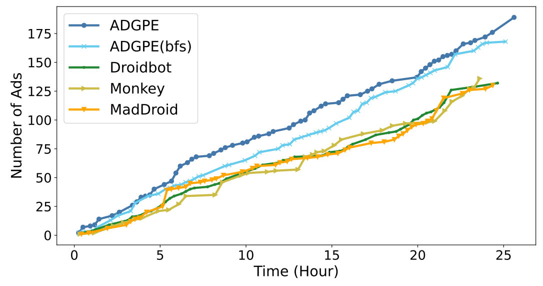


Fig. 9: Unique ads collected by time

VirusTotal in February, but later were reclassified as malware/PUAs in August. For instance, ADGPE detects a barcode scanner malware, "QR Scanner Rewards", promoted by several PUAs via ad libraries like Google AdMob and AppLovin. This malware exhibits behaviors identical to the malware that infected 10 million users, as documented in a 2021 security blog [87]. Additionally, the PUA named "Open Chat GBT - AI Chatbot App", released in February, was marked as benign but subsequently labeled as fleeceware in August as user reports of its fraudulent premium service increased. Notably, this app is promoted by one of the "lomol" trojans, which additionally promotes five malware and eight PUAs. *These findings show that app promotion ads have become a fertile ground to distribute PUAs/malware.*

D. RQ4: App Promotion Graph Construction

To evaluate the efficiency of ADGPE in constructing the app promotion graph, we compare ADGPE with general-purpose UI exploration approaches such as Monkey [32], DroidBot [31], and ADGPE implemented with breadth-first strategy (coined as ADGPE (bfs)). We also reproduce two SOTA ad detection approaches: MadDroid [8] and DARPA [33]. MadDroid uses breadth-first search to detect ads in WebView, ImageView, or ViewFlipper. DARPA leverages YOLO5 [88] to detect encouraging patterns of ads. Since DARPA lacks a UI exploration component, we manually labeled screenshots of app promotion ads from 30 apps within our affordable efforts and applied DARPA to detect ads within these images. Our comparisons use the same dataset from AndroZoo as the one used in Section III-C.

Ad Units. We further evaluate the effectiveness of each approach in collecting ads based on the diversity of ad types and the total number of ad units. Ad units are placeholders for developers to show ads. The content within an ad unit can change over time, but the number of ad units in an app is determined by the source code and thus remains constant [39].

TABLE VII: Comparison of ad coverage measured by collected ad units and ad types

Approaches	Ad Units	Ad Types		
		Inherent	Pop-up	Custom-Made
Droidbot [31]	76	27	38	9
Monkey [32]	71	26	34	11
DARPA [33]	8	8	0	0
MadDroid [8]	75	32	39	6
ADGPE (bfs)	131	52	58	15
ADGPE	**165**	**76**	**71**	**17**

TABLE VIII: Comparison of different link prediction models under two conditions

Approaches	w/o pre-trained embedding			pre-trained embedding		
	Hit@1	Hit@10	MRR	Hit@1	Hit@10	MRR
DistMult [73]	36.9	61.1	45.5	40.9	62.9	48.6
Complex [90]	37.8	60.0	45.4	33.8	59.1	42.4
Conve [91]	34.4	56.1	41.7	36.1	56.0	42.6
Path Inference	**58.2**	**66.6**	**61.3**	**59.5**	**67.4**	**62.4**

Additionally, we manually verify the number of ad units and the presence of each ad type in each app as ground truth.

The results in Table VII reveal that ADGPE finds the most ad units (165), which achieves 117.1% improvements over MadDroid, 132.4% over Monkey, and 26.0% over ADGPE (bfs). Besides inherent ads and pop-up ads, ADGPE also is more effective in finding custom-made ads (17), which are at least 54.5% better than MadDroid and Monkey.

Efficiency. Fig. 9 shows the number of unique app promotion ads collected over time by each approach. It can be seen that ADGPE significantly outperforms DroidBot and Monkey in collecting more ads with less time used, and performs slightly better than ADGPE (bfs). Specifically, ADGPE finds 54 ads within 10 hours, which is 20% more than ADGPE (bfs) (43 ads), 37% more than Droidbot (34 ads), 24% more than MadDroid, and 50% more than Monkey (27 ads).

App Promotion Ads in the Wild. We also conduct a case study on app promotion ads in the wild. These ads direct users outside Google Play and thus we need to manually download the promoted apps. Within our affordable efforts, we focus on a group of underground apps [89], as their ads and corresponding download links are usually hardcoded and thus easy to detect. Specifically, we adopt the same strategy to build the app promotion graph and start from two seed apps. We obtain a graph of 37 apps consisting of 21 PUAs/malware (5 gambling, 11 pornographic, 1 trojan, 4 adware). Interestingly, we notice that a scam gambling app can be reached by both seed apps, potentially revealing an interconnected underground profit network. *This finding demonstrates the capability of* ADGPE *ad-oriented UI exploration technique in detecting app promotion ads in the wild and its usability in studying real-world problems such as the underground economy.*

E. RQ5: App Promotion Reasoning

To evaluate the effectiveness of ADGPE in reasoning app promotion, we compare its performance with three SOTA link prediction models (DistMult [73], ComplEx [90], and ConvE [91]) across two conditions (with and without the pre-trained graph embeddings). We employ two widely used metrics as follows:

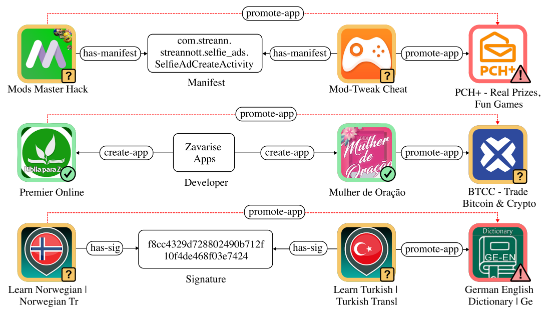
- *Hit@K* represents the proportion of times that the ground truth item appears in the top K output of the model.
- *Mean Reciprocal Rank (MRR)* calculates the average of the reciprocal of the ranks of the ground truth items, with higher scores indicating better performance.

Table VIII shows the results of our evaluation. First, we observe that the performance with pre-trained embedding is better than without pre-trained embedding, which shows that integrating app promotion relations into the entity representation captures more information about app promotion, thereby enabling better inference. Furthermore, Table VIII demonstrates that the path inference model outperforms the SOTA embedding models, achieving a Hit@1 of 59.5%, a Hit@10 of 67.4%, and an MRR of 62.4%. Notably, the path inference model secures a lead of 45.48% in Hit@1 and 28.40% in MRR relative to the best-performing SOTA model (i.e., DistMult).

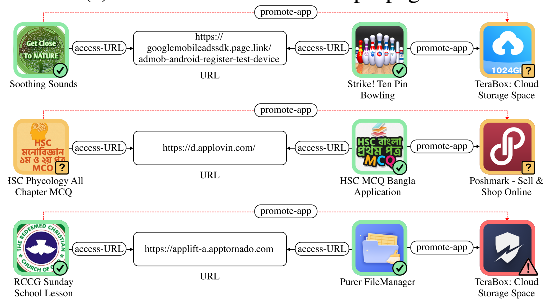
Aiding UI Exploration. To evaluate the effectiveness of the path inference model in the wild, we use the model to predict the app promotion links in the test set, which are masked in the training process. In the total 907 unique app promotion links, the model successfully predicts 318 of them (35.06%), indicating its potential to predict the promotion links that are not found by UI exploration. To further evaluate the effectiveness of our prediction results in facilitating our UI exploration, we measure the effort required to identify the 318 links. We do this by computing the average number of clicks made during the ad-oriented UI exploration using the recorded *clickTrace* for each link. On average, the ad-oriented UI exploration requires 18.3 clicks to identify a predicted link. Surprisingly, ADGPE's path inference model can even predict a link about an appwall ad by Google AdMob that requires 54 clicks to obtain. Given that the inference time of the path inference model is negligible compared to the resources and the time spent on UI exploration, these results demonstrate the model's effectiveness in augmenting ADGPE's ad-oriented UI exploration. *This demonstrates the promising direction of integrating AI to aid program analysis, consistent with previous work* [16], [53], [92], [93].

Uncovering Malware Promotion Mechanisms. We examine the inference paths and uncover two typical malware promotion mechanisms: *custom-made ad-based promotion mechanism*, which refers to app promotion ads hardcoded in the app's program, and *ad library-based promotion mechanism*, which relates to the app's interactions with ad servers. Fig. 10 illustrates these promotion mechanisms.

The custom-made ad-based promotion mechanism stems from shared developers, signatures, or manifest components among multiple apps. Such common factors lead the apps to exhibit similar code structures that tend to promote similar apps through app promotion ads. For instance, our model infers that a language learning app "Learn Norwegian-Norwegian Tr" promotes malware "German English Dictionary-Ge" because both "Learn Norwegian-Norwegian Tr" and "Learn Turkish-Turkish Transl", another



(a) Custom-made ad-based propagation



(b) Ad library-based propagation

Fig. 10: Examples of malware/PUA propagation mechanisms

language learning app, share the same signature and are developed by the same developer "lomol translator". As a result, both language learning apps share the same code structure related to app promotion and are inclined to promote the same apps.

The ad library-based promotion mechanism stems from the common URLs associated with ad libraries (e.g., AdMob) or ad promotion companies (e.g., Applovin) that are accessed by the apps. For example, a Sunday school learning app "RCCG Sunday School Lesson" and a file management app "Purer FileManager" access the same URL related to an ad promotion company AppTornado. Meanwhile, the "Purer FileManager" promotes a malicious cloud storage app "TeraBox: Cloud Storage Space." Based on such observation, the model concludes that "RCCG Sunday School Lesson" is highly likely to promote "TeraBox: Cloud Storage Space". This network promotion mechanism reveals the interconnected nature of app promotion graphs, as different apps would access the same ad server through common URLs. These findings indicate that the path inference model is effective in explaining the reason for malware promotion, thereby enhancing the explainability for malware detection.

Through analyzing the promotion paths of 950 verified malware/PUA in PIG, we found that the majority of advertiser apps were in the education (27.7%) and books (14.6%) categories. The majority of promoted malware/PUA apps were in the education (12.6%), entertainment (11.4%), and books (11.4%) categories. Among all these promotion paths, 697 paths (73.4%) used custom-made ads for promotion, while 253 paths (26.6%) used ad libraries for promotion.

VIII. Discussion

Implications. Our findings reveal a concerning result that many benign apps are directly or indirectly involved in the promotion of malware, indicating inadequate vetting performed by ad libraries and app markets. To address this, we advocate more rigorous vetting processes, stricter developer policies, and the adoption of open-source ad libraries [94]. Meanwhile, users should exercise caution and if necessary, employ security tools to vet the recommended apps before installation.

Limitations. The limitations of our app promotion graph construction are two-fold. First, our technique struggles with ads that appear after complex human interactions, such as ads that appear upon game failure [95]. We partially address this by expanding our seed app dataset. Second, false positives may arise, for instance, when a user is directed to Google Play for non-ad-related reasons like clicking a Facebook sharing button when Facebook is not installed. The main limitation of our path inference model is its lack of consideration for ad targeting, which some apps use to deliver personalized ads. While integrating personalized data like location and device information into our model could enhance its ability to detect and analyze such ads, emulating user profiles is a complex task and would require separate research efforts [14], [58].

Future Works. Our future works are three-fold. First, we plan to develop a unified mobile ads testing tool that uses static analysis [96], [97] to guide the UI exploration technique. This technique will further leverage large language models [98]–[100] to interact with the ads that require complex user interaction. Second, we plan to establish a long-term monitoring system, which allows us to track the changes and emerging trends in the app promotion ecosystem. This will also facilitate the app community to develop more effective countermeasures against potential threats. Additionally, we plan to explore the Chinese app promotion ecosystem which is distinct due to the absence of an official app market, reliance on its own ad libraries, and the common practice of directing users to developer-specific websites for app downloads.

IX. Related Work

Mobile Advertisements Identification. Previous research on mobile ads identification can be categorized into two types: identifying ad libraries, and identifying ad contents. For ad libraries identification, many studies compared the package names with a whitelist of collected ad libraries [101]–[103]. Recent works focused on using clustering-based methods to detect such libraries and achieve very high accuracy [6], [7]. However, ad content is dynamically generated and cannot be obtained via static analysis. Dynamic testing for UI exploration is a major approach for ad content identification [16]–[19], [104]. For example, MadDroid combined breadth-first UI exploration with rule-based HTTP hooking to extract ad content [8]. Researchers also leveraged computer vision techniques to detect button edges [15] or ad-related visual patterns [33] to recognize ad content. However, these techniques are biased to specific ad types, resulting low effectiveness in detecting app promotion ads. AdGPE gains insights from both our preliminary study and the design of existing techniques [8], [15], [33].

Malicious Advertisements Detection. The most pertinent previous research to our work is malicious mobile advertising. Existing studies analyzed deceptive ad contents [8] and malicious destinations (e.g., files, apps, and external websites) triggered by mobile ads [15]. However, these studies mainly examined the malicious behaviors of ad libraries, and none of them have specifically studied malware promotions through mobile ads. Another line of related work is graph-based Android malware detection. GNNs have gained prominence in this area due to their strong capabilities in representing structural data. APIGraph employed official Android API documentation to construct a relation graph and utilizes TransE [20] to learn the graph embeddings, which are subsequently used for API clustering and malware detection [22]. Both HinDroid [21] and Hawk [105] constructed a heterogeneous information network to model the relationship of malware based on API calls and app attributes. ADGPE diverges from these approaches by incorporating the app promotion relations. Moreover, the embedding trained by ADGPE serves more downstream tasks, such as promotion path inference.

Graph Path Inference. Reinforcement learning [106] (RL) is commonly used for path inference in graphs, treating the task as a Markov decision process. DeepPath [65] was the first RL-based model to find representative paths between entity pairs and employed a path ranking algorithm for training. Furthermore, MINERVA [76] learned how to guide the graph depending on the input entity-relation pair to find predictive paths where the second entity is unknown and must be acquired by inferring. Multi-Hop [64] proposed two improvements over MINERVA: reward shaping which uses soft rewards to capture triple semantics, and action dropout which enables more effective path exploration. We build our path inference model upon Multi-Hop to infer app promotion paths and explain malware promotion mechanisms.

X. Conclusion

We introduce a novel approach, ADGPE, that synergistically integrates app UI exploration with graph learning to automatically collect app promotion ads, detect malware promoted by these ads, and explain the malware promotion mechanisms employed by the detected malware. Our analysis of $18,627$ app promotion ads reveals a heightened risk for downloading apps via app promotion ads, which is hundreds of times higher than the likelihood of downloading malware from Google Play. Popular ad libraries are exploited by malicious developers to distribute a variety of malware. Our evaluations on real apps show that our malware detection model outperforms existing models in detecting ad-promoted malware, and our path inference model further reveals two primary malware promotion mechanisms. These findings demonstrate the effectiveness of combining dynamic program analysis with graph learning in studying malware promotion.

Acknowledgments

Xusheng Xiao's work is partially supported by the National Science Foundation under the grant CCF-2318483. Toby Jiajun Li's work was supported in part by an AnalytiXIN Faculty Fellowship, an NVIDIA Academic Hardware Grant, a Google Cloud Research Credit Award, a Google Research Scholar Award, a Google PSS Privacy Research Award, NSF Grant 2341187, and NSF Grant 2326378. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2014, pp. 221–233.

[2] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2015, pp. 89–103.

[3] "APP Lovin Success Stories," 2024. [Online]. Available: https://www.applovin.com/success-stories/

[4] Google AdMob, "Admob advantages," 2024. [Online]. Available: https://admob.google.com/home/admob-advantage/

[5] T. with Google, "How people discover, use, and stay engaged with apps," 2023. [Online]. Available: https://www.thinkwithgoogle.com/_qs/documents/331/how-users-discover-use-apps-google-research.pdf

[6] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: Fast and accurate detection of third-party libraries in android apps," in *Proceedings of the International Conference on Software Engineering Companion (ICSE-C)*, 2016, pp. 653–656.

[7] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "Libd: Scalable and precise third-party library detection in android markets," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*, 2017, pp. 335–346.

[8] T. Liu, H. Wang, L. Li, X. Luo, F. Dong, Y. Guo, L. Wang, T. Bissyandé, and J. Klein, "MadDroid: Characterizing and detecting devious ad contents for android apps," in *Proceedings of the ACM Web Conference (WWW)*, 2020, pp. 1715–1726.

[9] X. Gao, S. H. Tan, Z. Dong, and A. Roychoudhury, "Android testing via synthetic symbolic execution," in *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ICSE)*, 2018, pp. 419–429.

[10] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood, "Testing android apps through symbolic execution," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–5, 2012.

[11] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, "Guided, stochastic model-based GUI testing of android apps," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2017.

[12] T. Gu, C. Sun, X. Ma, C. Cao, C. Xu, Y. Yao, Q. Zhang, J. Lu, and Z. Su, "Practical GUI testing of android applications via model abstraction and refinement," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2019.

[13] D. Ran, Z. Li, C. Liu, W. Wang, W. Meng, X. Wu, H. Jin, J. Cui, X. Tang, and T. Xie, "Automated visual testing for mobile apps in an industrial seting," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2022, pp. 55–64.

[14] S. Nath, "Madscope: Characterizing mobile in-app targeted ads," in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2015, pp. 59–73.

[15] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. D. Riley, "Are these ads safe: Detecting hidden attacks through the mobile app-web interfaces," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[16] C. Hu and I. Neamtiu, "Automating gui testing for android applications," in *Proceedings of the International Workshop on Automation of Software Test (AST)*, 2011, pp. 77–83.

[17] S. Nath, F. X. Lin, L. Ravindranath, and J. Padhye, "Smartads: bringing contextual ads to mobile apps," in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013, pp. 111–124.

[18] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: Detecting and characterizing ad fraud in mobile apps," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 57–70.

[19] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps," in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2014, pp. 204–217.

[20] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, vol. 26, 2013.

[21] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2017, pp. 1507–1515.

[22] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020, pp. 757–770.

[23] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[24] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, vol. 30, 2017.

[25] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[26] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," *arXiv preprint arXiv:1905.12265*, 2019.

[27] T. K. Ho, "Random decision forests," in *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)*, 1995, pp. 278–282.

[28] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–34, 2019.

[29] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019, pp. 47–62.

[30] Y. Shen and G. Stringhini, "Andruspex: leveraging graph representation learning to predict harmful app installations on mobile devices," in *Proceedings of the European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 562–577.

[31] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: a lightweight ui-guided test input generator for android," in *Proceedings of the IEEE/ACM International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 23–26.

[32] Google, "UI/Application Exerciser Monkey," 2023. [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey

[33] Z. Cai, Y. Nan, X. Wang, M. Long, Q. Ou, M. Yang, and Z. Zheng, "DARPA: Combating asymmetric dark ui patterns on android with run-time view decorator," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023, pp. 480–493.

[34] G. A. H. Center, " How the Google Ads auction works," 2023. [Online]. Available: https://support.google.com/google-ads/answer/6366577

[35] Facebook, "Facebook ads: About ad auctions," 2023. [Online]. Available: https://www.facebook.com/business/help/430291176997542

[36] G. Help, "Google Ads Help Center: About App campaigns," 2023. [Online]. Available: https://support.google.com/google-ads/answer/6247380.

[37] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, p. 356–367.

[38] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on android," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, p. 2187–2200.

[39] G. A. Help, "Ad units, ad formats, & ad types," 2023. [Online]. Available: https://support.google.com/admob/answer/6128738?hl=en

[40] M. B. H. Center, "Types of Meta ad formats," 2023. [Online]. Available: https://www.facebook.com/business/help/1263626780415224?id=802745156580214

[41] A. Support, "Ad Formats," 2023. [Online]. Available: https://support.applovin.com/hc/en-us/articles/11259746423565-Ad-Formats

[42] "Google AdMob: Get Started," 2024. [Online]. Available: https://developers.google.com/admob/android/quick-start

[43] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet?(e)," in *Proceeding of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 429–440.

[44] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability (ToR)*, vol. 68, no. 1, pp. 45–66, 2018.

[45] Y. Ma, Y. Huang, Z. Hu, X. Xiao, and X. Liu, "Paladin: Automated generation of reproducible test cases for android apps," in *Proceedings of the International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2019.

[46] Appium, 2023. [Online]. Available: https://appium.io

[47] G. for Developers, "UI Automator," 2023. [Online]. Available: https://developer.android.com/training/testing/other-components/ui-automator.

[48] S. Son, D. Kim, and V. Shmatikov, "What mobile ads know about mobile users." in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[49] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it:ui state inference and novel android attacks," in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2014, pp. 1037–1052.

[50] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2012, pp. 95–109.

[51] M. D. Ernst, R. Just, S. Millstein, W. Dietl, S. Pernsteiner, F. Roesner, K. Koscher, P. Barros, R. Bhoraskar, S. Han, P. Vines, and E. X. Wu, "Collaborative verification of information flow for a high-assurance app store," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014, pp. 1092–1104.

[52] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 303–313.

[53] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, , and J. Lu, "DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[54] K. Xu, Y. Li, R. H. Deng, and K. Chen, "DeepRefiner: Multi-layer android malware detection system applying deep neural networks," in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[55] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the IEEE/ACM Working Conference on Mining Software Repositories (MSR)*, 2016, pp. 468–471.

[56] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2017, pp. 845–854.

[57] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[58] T. Book and D. S. Wallach, "An empirical study of mobile ad targeting," *arXiv preprint arXiv:1502.06577*, 2015.

[59] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft, "Breaking for commercials: characterizing mobile advertising," in *Proceedings of the Internet Measurement Conference (IMC)*, 2012, pp. 343–356.

[60] R. Shao, V. Rastogi, Y. Chen, X. Pan, G. Guo, S. Zou, and R. Riley, "Understanding in-app ads and detecting hidden attacks through the mobile app-web interface," *IEEE Transactions on Mobile Computing (TMC)*, vol. 17, no. 11, pp. 2675–2688, 2018.

[61] A. Desnos and G. Gueguen, "Androguard documentation," 2018. [Online]. Available: ObtenidodeAndroguard:https://androguard.readthedocs.io/en/latest

[62] A. Kao and S. R. Poteet, *Natural language processing and text mining*. Springer Science & Business Media, 2007.

[63] J. Gerlings, A. Shollo, and I. Constantiou, "Reviewing the need for explainable artificial intelligence (xai)," *arXiv preprint arXiv:2012.01007*, 2020.

[64] X. V. Lin, R. Socher, and C. Xiong, "Multi-hop knowledge graph reasoning with reward shaping," in *Proceedings of the Conference on*

Empirical Methods in Natural Language Processing (EMNLP), 2018, pp. 3243–3253.

[65] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Reinforcement learning*, pp. 5–32, 1992.

[66] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[67] Y. Ishii, T. Watanabe, F. Kanei, Y. Takata, E. Shioji, M. Akiyama, T. Yagi, B. Sun, and T. Mori, "Understanding the security management of global third-party android marketplaces," in *Proceedings of the ACM SIGSOFT International Workshop on App Market Analytics (WAMA)*, 2017, pp. 12–18.

[68] A. Cortesi, M. Hils, T. Kriechbaumer, and contributors, "mitmproxy: A free and open source interactive HTTPS proxy," 2023. [Online]. Available: https://mitmproxy.org/

[69] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[70] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[71] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bring order to the web," technical report, Stanford University, Tech. Rep., 1998.

[72] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 249–256.

[73] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[74] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[75] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2020, pp. 4116–4126.

[76] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[77] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[78] Y. Zhao, T. Liu, H. Wang, Y. Liu, J. Grundy, and L. Li, "Are mobile advertisements in compliance with app's age group?" in *Proceedings of the ACM Web Conference (WWW)*, 2023, pp. 3132–3141.

[79] Google Play, "The magnet searcher," 2024. [Online]. Available: https://play.google.com/store/apps/details?id=cili.niao.search.bt.ci.li

[80] The distribution site, "The malicious downloader promoted by the magnet searcher," 2024. [Online]. Available: https://apk.apkdownqd6.top/

[81] N. Zhong and F. Michahelles, "Google play is not a long tail market: An empirical analysis of app adoption on the google play app market," in *Proceedings of the Annual ACM Symposium on Applied Computing (SAC)*, 2013, pp. 499–504.

[82] Yahoo, "Android apps with 2.5 million downloads are showing ads with display off — delete these now," 2023. [Online]. Available: https://www.yahoo.com/lifestyle/android-apps-2-5-million-192411303.html

[83] Google Play, "The trojan disguising as a Chinese dictionary," 2024. [Online]. Available: https://play.google.com/store/apps/details?id=com.qiushui.android.app.chdir

[84] AppsGeyser, "No code app maker: AppsGeyser," 2024. [Online]. Available: https://appsgeyser.com/

[85] McAfee, "Similar rogue security softwares reported by McAfee," 2024. [Online]. Available: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/new-hiddenads-malware-that-runs-automatically-and-hides-on-google-play-1m-users-affected/

[86] Facebook, "The Facebook profile of "Fancy Battery: Cleaner, Secure"," 2024. [Online]. Available: https://m.facebook.com/p/Fancy-Battery-100070380125569/

[87] Malwarebytes, "Same malicious behaviours reported by Malwarebytes," 2024. [Online]. Available: https://www.malwarebytes.com/blog/news/2021/02/barcode-scanner-app-on-google-play-infects-10-million-users-with-one-update

[88] G. Jocher, "YOLOv5 by Ultralytics," May 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[89] Z. Chen, J. Liu, Y. Hu, L. Wu, Y. Zhou, Y. He, X. Liao, K. Wang, J. Li, and Z. Qin, "Deuedroid: Detecting underground economy apps based on utg similarity," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 223–235.

[90] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 2071–2080.

[91] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 32, no. 1, 2018.

[92] C. Liu, H. Wang, T. Liu, D. Gu, Y. Ma, H. Wang, and X. Xiao, "Promal: precise window transition graphs for android via synergy of program analysis and machine learning," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2022, pp. 1755–1767.

[93] D. Adamo, M. K. Khan, S. Koppula, and R. Bryce, "Reinforcement learning for android gui testing," in *Proceedings of the ACM SIGSOFT International Workshop on Automating Test Case Design, Selection, and Evaluation (A-Test)*, 2018, pp. 2–8.

[94] "Ethical Ad Server," 2024. [Online]. Available: https://ethical-ad-server.readthedocs.io/en/latest/.

[95] G. Xu, Y. Hu, Q. Guo, R. He, L. Li, G. Xu, Z. Han, and H. Wang, "Dissecting mobile offerwall advertisements: An explorative study," in *Proceedings of the International Conference on Software Quality, Reliability, and Security (QRS)*, 2020, pp. 518–526.

[96] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "Iconintent: automatic identification of sensitive ui widgets based on icon classification for android apps," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2019, pp. 257–268.

[97] A. Rountev and D. Yan, "Static reference analysis for gui objects in android software," in *Proceedings of the Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2014, pp. 143–153.

[98] OpenAI, "Chatgpt: Applications, opportunities, and threats," *arXiv preprint arXiv:2304.09103*, 2023.

[99] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, vol. 33, pp. 1877–1901, 2020.

[100] Y. Wang, Y. Zhang, Y. Li, and X. Liu, "A bibliometric review of large language models research from 2017 to 2023," *arXiv preprint arXiv:2304.02020*, 2023.

[101] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2012, pp. 101–112.

[102] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of android ad library permissions," *arXiv preprint arXiv:1303.0857*, 2013.

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

[104] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: automatic security analysis of smartphone applications," in *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013, pp. 209–220.

[105] Y. Hei, R. Yang, H. Peng, L. Wang, X. Xu, J. Liu, H. Liu, J. Xu, and L. Sun, "Hawk: Rapid android malware detection through heterogeneous graph attention networks," *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, pp. 4703–4717, 2021.

[106] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research (JAIR)*, pp. 237–285, 1996.