DPSCAN: Knowledge-Augmented LLM-Driven Detection of Dark Patterns in Mobile Ads

Yuxuan Shang* — *Zhejiang University* — shangyuxuan@zju.edu.cn
Haitao Xu* — *Zhejiang University* — haitaoxu@zju.edu.cn
Ruikang Xu — *Zhejiang University* — 3220105566@zju.edu.cn
Xingming Chen — *Zhejiang University* — 3230102275@zju.edu.cn
Shengdi Jin — *Zhejiang University* — 3230103902@zju.edu.cn
Shuai Hao — *Old Dominion University* — shao@odu.edu
Chuan Yue — *Colorado School of Mines* — chuanyue@mines.edu
Wenrui Ma — *Zhejiang Gongshang University* — mawenrui@zjgsu.edu.cn
Zhao Li — *Hangzhou Yugu Technology* — lzjoey@gmail.com

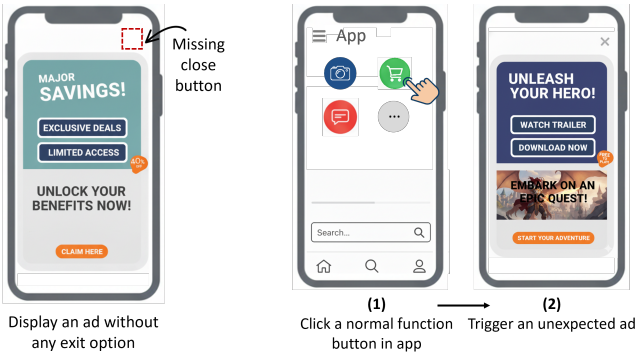
***Abstract*—Mobile ads generate substantial revenue but are increasingly accompanied by deceptive user interfaces (UIs) known as dark patterns, which manipulate user autonomy and cause harms ranging from frustration to security vulnerabilities. Existing auditing tools are confined to static, single-screen analysis, overlooking dynamic dark patterns that span multiple UIs and user interactions. Moreover, no existing tool automates the exploration needed to trigger such patterns. To address these gaps, we first conduct a systematic study of a large video dataset documenting mobile ad-related dark pattern instances and human auditors' strategies for triggering and identifying them. From this analysis, we distill heuristic exploration strategies and propose sequential representations for ten common dark pattern categories, each modeled as a UI-interaction chain from a starting UI to a landing UI. Based on these insights, we develop DPSCAN, an LLM-driven exploration system for efficient discovery of both static and dynamic ad-related dark patterns in mobile apps. Specifically, DPSCAN accurately represents UIs using screenshots and view hierarchies augmented with computer vision, employs retrieval-augmented generation to provide similar UI exemplars, and guides an LLM to comprehend the UI and select exploration strategies by referencing relevant human auditors' decision-making cases, reporting a dark pattern whenever a matching sequential representation is observed. Evaluation on a verified dataset shows that DPSCAN achieves near-human performance, delivering an F1-score of 96.76% and comprehensive exploration (recalling 91.71% of dark patterns compared to manual efforts), outperforming state-of-the-art detectors DPGuard and UIGuard by 20.36% and 55.56%, respectively. Furthermore, a large-scale evaluation of 2,048 real-world popular apps reveals that 18.26% of apps and 35.81% of displayed ads exhibit dark patterns, with the highest prevalence (33.92%) observed in apps rated for teens. DPSCAN demonstrates strong robustness against adversarial attacks, offering a practical solution for large-scale dark pattern auditing, benefiting auditors (*e.g.*, app stores and lawmakers), researchers, and app users. Our implementation and datasets are available at https://github.com/syx1031/DPScan.**

*These authors contributed equally to this work.

Network and Distributed System Security (NDSS) Symposium 2027
22-26 March 2027, Seoul, Korea
ISBN 978-1-970672-09-1
https://dx.doi.org/10.14722/ndss.2027.230627
www.ndss-symposium.org

I. INTRODUCTION

Nowadays, mobile advertisements generate substantial revenue for app developers, representing a global market valued at $262.84 billion in 2025 and growing rapidly [1]. To maximize profits, developers increasingly employ deceptive user interfaces (UIs) to manipulate user autonomy and influence their decisions. For example, they may deliberately omit close buttons to prevent users from dismissing an ad (Fig. 1(a)), or display unexpected ads after users click functional buttons, thereby interfering with app functions (Fig. 1(b)). This design paradigm is termed "dark patterns" by the Federal Trade Commission [2]. Dark patterns inflict a broad spectrum of harms, ranging from psychological frustration [3] and economic losses [4] to severe security and privacy vulnerabilities [5], [6].



(a) Ad Without Exit Option (b) Unexpected Full-Screen Ads

Fig. 1: Cases of two dark patterns within mobile ads

Given these harmful effects, auditors such as app stores and lawmakers have prohibited such practices [7], [8], creating a pressing demand for large-scale auditing of dark patterns within mobile apps. However, existing auditing tools have notable limitations in accuracy and usability. One paradigm, exemplified by UIGuard [9] and AidUI [10], relies on fixed heuristic rules that detect dark patterns by identifying specific textual patterns (*e.g.*, "`promoted`" or "`ads`") or icons (*e.g.*, ▷) on a UI page. Such approaches fail to detect many dark pattern instances that use novel textual or visual patterns (*e.g.*, "`videos`" rather than "`ads`", "AD" rather than "▷"). Recent advances such as

DPGuard [5] and AutoBot [11] partially address this limitation by leveraging large language models (LLMs) to comprehend UIs and assess whether their semantics align with dark pattern definitions. Nevertheless, these methods remain confined to static patterns within a single UI, overlooking dynamic patterns that span multiple UIs, involve user interactions on specific UI elements (*e.g.*, unexpected full-screen ads in Fig. 1), and are prevalent in mobile apps.

Given the limitations of existing methods, there is an urgent need for tools that support large-scale auditing of dark patterns in mobile ads (hereafter "ads dark patterns" [12]). Specifically, auditors require an end-to-end system that automates app installation, execution, UI navigation, and dark pattern identification. Achieving accurate detection is nontrivial, with two main challenges: how to efficiently explore apps to trigger dark patterns, and how to accurately identify instances within heterogeneous UIs. Therefore, such a tool must satisfy three requirements: (1) effectively leverage human auditors' strategies for triggering and detecting dark patterns; (2) conduct efficient and extensive UI exploration focused on triggering dark patterns; (3) enable accurate identification of dark patterns across diverse real-world UIs.

To understand auditors' strategies, we systematically study AdsDP [12], the most comprehensive video dataset documenting numerous ads dark pattern instances and how auditors attempt to trigger them. From this analysis, we identify categories of UIs (*e.g.*, purchase UI), UI elements (*e.g.*, close button), and user interactions (*e.g.*, click) involved in the occurrence of ads dark patterns. Consequently, we propose sequential representations for the ten most common categories of ads dark patterns. Each representation consists of a sequence of interactions from a starting UI to a landing UI, reflecting real-world observable manifestations. A manually verified experiment demonstrates the accuracy and robustness of these sequences. We also observe heuristic exploration strategies that auditors perform when trying to trigger dark patterns, such as navigating to the purchase page for ad removal services.

Based on these insights, we introduce DPSCAN, a knowledge-augmented and LLM-driven system for exploring apps and detecting dark patterns in mobile ads. DPSCAN works end-to-end, taking an app as input and outputting detected ads dark patterns within a dynamic test period. Specifically, given a UI page, DPSCAN obtains both visual and structural information (namely UI screenshots and view hierarchies), augmented with a vision-based model for interactive apps built on engines like Unity [13] whose certain UI elements (*e.g.*, close buttons) are not exposed in the standard view hierarchy. It then retrieves similar UI elements from a retrieval-augmented generation (RAG) [14], [15] repository containing typical UI elements and their textual and visual content, providing useful references for element categorization. Such information is fed to an LLM to comprehend the UI elements in terms of their type, potential function, boundaries, and coordinates. Based on the current UI understanding, DPSCAN instructs the LLM to select the most promising heuristic exploration strategies toward discovering ads dark patterns, providing relevant human auditors' decision-making cases with similar UI context as exemplars. Throughout exploration, DPSCAN reports a dark pattern upon observing a matching sequential representation.

We evaluate DPSCAN on 97 apps derived from AdsDP [12]. It achieves accurate detection across all categories, with an average precision of 96.65%, recall of 96.86%, and F1 of 96.76%. Compared to manual efforts, it demonstrates comprehensive UI navigation capabilities, successfully triggering 91.71% of the manually identified dark pattern categories. An ablation study further confirms the necessity of its key components. In addition, DPSCAN outperforms state-of-the-art detectors, surpassing DPGuard [5] and UIGuard [9] in average F1 by 20.36% and 55.56%, respectively. Leveraging DPSCAN, we conduct a large-scale empirical study on 2,048 popular real-world apps, revealing dark patterns in 18.26% of apps and 35.81% of ads, with the highest prevalence in *Games* (44.49%) and teen-rated (12+) apps (33.92%). Finally, DPSCAN shows strong robustness against adversarial attacks, achieving only a 7.50% bypass rate on 40 crafted attempts.

In summary, our contributions are as follows:

- We systematically investigate human auditors' strategies for discovering mobile ads dark patterns, derive sequential representations for ten common categories, and construct an RAG repository of relevant UIs, elements, and interactions.
- We develop DPSCAN, a knowledge-augmented, LLM-driven system that automates the exploration and detection of mobile ads dark patterns by integrating human auditing expertise.
- We extensively evaluate DPSCAN on verified and real-world datasets, demonstrating its effectiveness and robustness for large-scale dark pattern auditing.

II. Background and Related Work

We introduce the background of ads dark patterns and review prior work on their categorization and detection.

A. Ads Dark Patterns

Existing literature has identified various ad-related dark patterns ("ads dark patterns" [12]). Brignull *et al.* [16] presented the first work on dark patterns and introduced "disguised ads," where ads mimic regular content to deceive users. Di Geronimo *et al.* [18] manually explored 240 popular Android apps and uncovered three novel ads dark patterns, such as requiring users to watch ads to unlock app features. Gao *et al.* [19] analyzed user feedback and highlighted prevalent negative phenomena, including excessive ad quantity, overly long durations, and oversized ads. Aligning with these findings, Google Play has banned dark pattern practices in its recent policies [7], [8], including injecting ads at unexpected moments or tying app functionality to ad display.

Building on this, Shang *et al.* [12] conducted a large-scale empirical study dedicated to ads dark patterns in mobile apps. By manually operating 485 iOS apps on real devices, recording all operations, and annotating dark patterns, they proposed a taxonomy of 15 ads dark pattern types and publicly released *AdsDP* [20], the largest video dataset of ads dark patterns, documenting how human researchers trigger various

TABLE I: Summary of existing dark pattern taxonomies, highlighting the evolution from generic to mobile ad-focused ones.

Studies	Brignull *et al.* [16]	Gray *et al.* [17]	Di Geronimo *et al.* [18]	Mansur *et al.* [10]	Chen *et al.* [9]	Shi *et al.* [5]	Shang *et al.* [12]
Platforms	Web	Mobile	Mobile	Mobile & Web	Mobile	Mobile & Web	Mobile
# Dark Pattern Categories	14	12	16	10	15	21	15
# Ads Dark Pattern Categories	1	1	4	1	4	6	15
# Dark Pattern Instances	57	112	1,787	301	1,660	7,044	5,782
Year	2010	2018	2020	2023	2023	2025	2025

patterns. The dataset comprises 718 videos totaling 60 hours, with fine-grained annotations of 5,782 instances across the 15 categories. A *manual* was also released, documenting the standardized workflow for dark pattern exploration and identification, specifying relevant UIs, interaction types, and resulting dark patterns.

According to *AdsDP*, ads dark patterns can be categorized as *static* vs. *dynamic* (based on dynamism) and *ad-level* vs. *app-level* (based on whether the pattern occurs within an ad or elsewhere in the app). We adopt these concepts in this study.

Static vs. dynamic. Static patterns are identifiable from a single UI screenshot (*e.g.*, `Ad without Exit Option` in Fig. 1(a)); dynamic patterns require analyzing multiple sequential UIs triggered by user interactions (*e.g.*, `Unexpected Full-Screen Ads` in Fig. 1(b)).

Ad-level vs. app-level. Ad-level patterns occur specifically within ad displays (*e.g.*, `Ad without Exit Option`); app-level patterns persist throughout general app usage (*e.g.*, `Unexpected Full-Screen Ads`).

B. Dark Pattern Taxonomies

Beyond ads dark patterns, generic dark patterns in UI design have been identified across various platforms and domains. Since Brignull *et al.* [16] introduced the concept over a decade ago, researchers have proposed taxonomies for dark patterns observed on different platforms (*e.g.*, mobile apps [9], [10], [12], [17], websites [4], [11], and IoT devices [21]) and in different contexts (*e.g.*, e-commerce [4], social media [22], [23], games [24], cookie banners [25]–[28], and virtual reality [29]). Gray *et al.* [17] expanded Brignull's taxonomy, categorizing mobile dark patterns into five themes (*e.g.*, `Nagging`, `Sneaking`) based on developers' intentions. Gunawan *et al.* [30] revealed discrepancies in dark pattern manifestation across mobile apps, mobile browsers, and desktop websites. More recently, Shi *et al.* [5] refined previous taxonomies by integrating security and privacy perspectives. As summarized in Table I, early work focused on web with few ad-related categories; subsequent studies expanded to mobile, with Di Geronimo *et al.* [18] and Chen *et al.* [9] identifying 16 and 15 categories from 1,787 and 1,660 instances, respectively. Mansur *et al.* [10] and Shi *et al.* [5] covered both mobile and web, the latter reporting 21 categories and 7,044 instances. Notably, Shang *et al.* [12] (the *AdsDP* work) provide the most comprehensive ad-specific taxonomy, listing 15 ad-related categories from 5,782 instances within mobile apps. This evolution illustrates growing granularity and increasing emphasis on mobile advertising contexts over time.

C. Dark Pattern Detection

Prior work on automated dark pattern detection can be broadly categorized into rule-based heuristics, probabilistic methods, and LLM-based approaches.

(1) Rule-based and heuristic methods. Mathur *et al.* [4] crawled 11k shopping websites and identified dark patterns using an unsupervised clustering algorithm to group similar texts. UIGuard [9], AidUI [10], and DarkDialogs [31] focus on identifying specific visual designs and fixed textual expressions within single UI screenshots. These works leverage computer vision and natural language processing to extract UI components (*e.g.*, icons, buttons, text) and apply heuristic rules summarized by experts. For example, the co-occurrence of "`accept ... privacy policy`" text with a pre-selected checkbox may be flagged as the dark pattern "Preselection." These tools pioneered detection on mobile app UIs but cannot handle elements with unseen patterns, limiting their generalizability to ads dark pattern detection due to rapidly evolving UI designs.

(2) Probabilistic model-based methods. Tung *et al.* [32] trained a Naïve Bayes classifier on the dataset of Mathur *et al.* [4] to detect dark patterns on websites. Soe *et al.* [33] employed a Gradient Boosted Tree classifier for cookie banners. However, these approaches are restricted to textual patterns and neglect broader webpage context, leading to many false positives [9].

(3) LLM-based detection. DPGuard [5] feeds raw screenshots to an LLM to assess whether their semantics align with dark pattern definitions. AutoBot [11] combines vision models with LLMs: it extracts UI element coordinates and text, then uses an LLM to understand contextual semantics. Schäfer *et al.* [34] used GPT-4o to directly rectify dark patterns in webpage source code. Despite promising performance, these methods overlook dynamic dark patterns prevalent in ad services. More critically, they cannot perform end-to-end detection with autonomous UI navigation and dark pattern triggering, thus do not scale to large-scale in-app auditing.

Summary of limitations. Across all existing work, the following limitations persist: (1) Most methods (*e.g.*, UIGuard [9], AidUI [10], DPGuard [5], AutoBot [11]) operate on static, single-screen UIs, thereby overlooking dynamic dark patterns that span multiple screens and require sequential user interactions—a gap explicitly noted by Shang *et al.* [12]. (2) None of the prior work automates the exploration needed to trigger dark patterns within mobile apps; all rely on manual or scripted navigation, which does not scale. Consequently, no existing solution supports end-to-end, large-scale auditing of ads dark patterns in mobile apps.

III. Human Auditors' Strategies for Dark Pattern Detection

From the 15 ad-related dark pattern categories listed in AdsDP [12], we excluded those with too few instances (fewer than eight) to ensure sufficient statistical support for deriving general strategies, yielding ten types as our subjects (Table II).

TABLE II: Taxonomy of ten dark pattern categories analyzed, including theme, description, dynamism, and occurrence level.

Theme	Dark Pattern	Description	Dynamism	Level	S&P Threats
Nagging	App Resumption Ads	Users return to the home screen temporarily. Upon switching back to the app, they are forced to watch an ad.	Dynamic	App-level	User control hijacking
	Unexpected Full-Screen Ads	A full-screen ad unexpectedly appears when clicking a button, or appears unprompted without any user interaction.	Dynamic	App-level	User control hijacking
	Auto-Redirect Ads	Ad redirects to its landing page without the user clicking a "Skip" or "Close" button.	Dynamic	Ad-level	User control hijacking
Forced Action	Paid Ad Removal	App offers paid options to remove ads.	Static	App-level	Excessive privacy disclosure
	Reward-Based Ads	App requires users to watch ads for benefits.	Static	App-level	Excessive privacy disclosure
Obstruction	Ad Without Exit Option	Ad does not provide a close button or delays its appearance during display.	Static	Ad-level	Autonomy compromise
	Ad Closure Failure	Ad does not close after clicking the close button.	Dynamic	Ad-level	User control hijacking
Interface Interference	Multiple Close Buttons	Ad shows multiple close buttons simultaneously.	Static	Ad-level	Autonomy compromise
	Bias-Driven UI Ads	Ad presents options to users and prominently highlights the option benefiting the advertiser.	Static	Ad-level	Autonomy compromise
	Disguised Ads	Ad visually resembles normal content in apps or system UI.	Static	Ad-level	Autonomy compromise

These patterns span four themes: *Nagging*, *Forced Action*, *Obstruction*, and *Interface Interference*. Under *Nagging*, `App Resumption Ads`, `Unexpected Full-Screen Ads`, and `Auto-Redirect Ads` describe ads that appear when users return to the app, unexpectedly occupy the full screen, or redirect users without interaction. *Forced Action* includes `Paid Ad Removal` and `Reward-Based Ads`, which require users to pay to remove ads or watch ads for benefits. *Obstruction* includes `Ad Without Exit Option` and `Ad Closure Failure`, where ads lack a close button or fail to close upon user interaction. Under *Interface Interference*, `Multiple Close Buttons`, `Bias-Driven UI Ads`, and `Disguised Ads` describe ads that present multiple close options (both genuine and deceptive), visually emphasize advertiser-benefiting options (*e.g.*, purchase or download), or mimic normal in-app content or iOS system UI. Overall, four patterns are dynamic, and six are static, with an equal split between app-level and ad-level occurrences.

These ad-related dark patterns directly correspond to practices prohibited by major app store policies [7], [8] and pose concrete security and privacy threats across three dimensions: user control hijacking, excessive privacy disclosure, and autonomy compromise. *User control hijacking*, exemplified by `App Resumption Ads`, `Unexpected Full-Screen Ads`, `Auto-Redirect Ads`, and `Ad Closure Failure`, redirects users to unintended ads containing trackers or triggers unauthorized transitions to external apps. *Excessive privacy disclosure*, associated with `Paid Ad Removal` and `Reward-Based Ads`, often relies on opaque policies that encourage users to disclose sensitive information. *Autonomy compromise*, arising from `Ad Without Exit Option`, `Multiple Close Buttons`, `Bias-Driven UI Ads`, and `Disguised Ads`, manipulates visual designs to degrade user vigilance and compel users to select advertiser-favoring options. These threats—ranging from forced redirects to opaque data collection—underscore the need to detect such deceptive practices.

One main challenge for accurate dark pattern detection is the ambiguity of determination standards. A promising solution is to leverage human auditors' expertise, largely overlooked in prior research but well documented in AdsDP through video recordings of ads dark pattern instances and rich narratives of how auditors attempt to trigger various dark patterns. Therefore, we conducted a thorough investigation of AdsDP to answer two critical questions: (1) how human auditors efficiently explore unseen apps to trigger dark patterns, and (2) how to accurately identify these instances amid heterogeneous UIs.

A. Exploration Heuristics for Triggering Dark Patterns

Unlike general app exploration that prioritizes UI coverage, dark pattern exploration focuses on identifying clickable elements most likely to lead to UI pages where dark patterns may reside. The latter relies heavily on human intuition or past experience with general app usage.

Our four researchers independently reviewed the AdsDP *manual* and, through discussion, extracted and consolidated recurring exploration heuristics. They observed that human auditors first conduct broad UI exploration toward UIs most likely to harbor dark patterns (*e.g.*, in-app purchase interfaces). Upon reaching such UI pages, auditors typically perform the following five attempts to trigger dark pattern instances. Fig. 2 illustrates example attempts by AdsDP human researchers on the app "Oxford Dictionary."

- **Switching between an in-app UI and the smartphone home screen** (Fig. 2(a)): Inside an app's UI page, human auditors tap the Home Indicator to switch from the app to the home screen, and after a while tap the app icon to return to the app while checking whether unexpected full-screen ads suddenly appear. According to the AdsDP manual, the `App Resumption Ads` dark pattern may appear.
- **Navigating to the purchase page for ad removal services**: Based on their past app usage experience, human auditors tap the crown icon in the top-right corner to navigate to the purchase page for ad removal services, representing an instance of `Paid Ad Removal`.

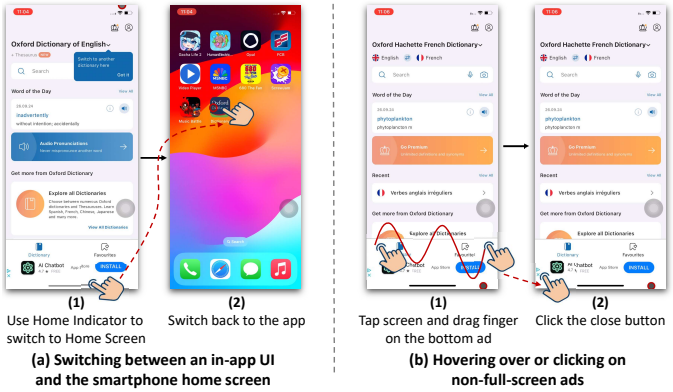


Fig. 2: Example heuristics to trigger dark patterns

- **Attempting to dismiss full-screen ads**: On a full-screen ad, human auditors locate the close button and attempt to click it to dismiss the ad. This operation may reveal several dark patterns, including `Ad Closure Failure`, `Ad Without Exit Option`, `Multiple Close Buttons`, `Auto-Redirect Ads`, `Bias-Driven UI Ads`, or `Disguised Ads`.
- **Hovering over or clicking on non-full-screen ads**: Human auditors hover over or click on a non-full-screen ad to observe if a landing page is triggered, and eventually dismiss the ad by clicking the close button (illustrated in Fig. 2(b)). Several dark patterns may appear, including `Ad Closure Failure`, `Ad Without Exit Option`, `Multiple Close Buttons`, `Bias-Driven UI Ads`, or `Disguised Ads`.
- **Exploring as many app functionalities as possible**: When no other priority operations remain, human auditors persistently explore the app's features and interfaces, covering as many functionalities as possible. `Unexpected Full-Screen Ads`, `Reward-Based Ads`, or `Paid Ad Removal` may be triggered.

B. Identifying Dark Patterns via UI-Interaction Traces

The occurrence of ads dark patterns is typically a dynamic process that may span multiple UI pages and require specific triggering conditions. Therefore, clearly defining what constitutes a dark pattern is crucial for accurate identification.

According to video recordings of dark pattern appearances from AdsDP, we found that the dynamic process typically involves a *starting UI*, specific user interactions (*e.g.*, click or hover) on UI elements, and then a *landing UI*.

1) UIs, Elements, and User Interactions Involved in Ads Dark Pattern: We summarize the UIs, UI elements, and user interactions involved in ads dark patterns as follows.

(1) UI Pages Constituting a Dark Pattern. The following five categories of UI pages may form part of a dark pattern process, with cases shown in Fig. 3:

- **System UI** (3(a)) refers to the UIs belonging to the mobile operating system rather than any app. Typical examples include Home Screen and Control Center.
- **App Functional UI** refers to UIs that facilitate an app's main functions. Examples include the app's homepage, settings page, and UIs closely related to main functions (*e.g.*, product showcases in shopping apps, gameplay screens in game apps).
- **Purchase UI** (3(b)) refers to UIs dedicated to real-money exchanges for virtual goods or services, typically handling purchases of premium memberships, virtual currency, or ad-removal subscriptions.
- **Full-Screen Ad Interface** refers to a single ad occupying the entire or vast majority of the screen, suspending user interaction with the host app.
- **Ad Landing Page** (3(c)) refers to the UI to which a user is redirected after clicking an ad. Examples include the download page of the promoted app in an app store or the advertiser's official website.

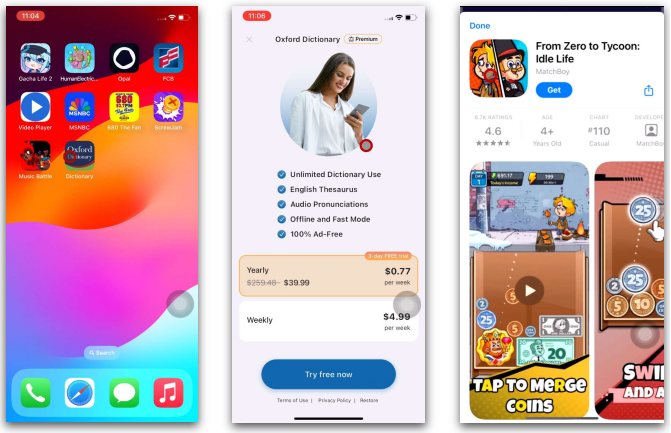


Fig. 3: Instances of UIs related to ads dark patterns

(2) Elements Possibly Part of a Dark Pattern Representation. We identified the following seven categories of UI elements possibly related to dark patterns. UI elements here broadly refer to images, text, icons, buttons, and all semantically independent UI components observable on the current UI page, typically bounded as a rectangle (x, y, w, h).

- **Non-Full-Screen Ad**: Ads typically displayed as banners in a corner or embedded within regular UI content.
- **Close Button**: Elements that dismiss a UI component (*e.g.*, an ad or pop-up) when clicked, typically contain symbols "×" or text like "Continue to app."
- **Ad Removal Element**: A paid service to remove ads from the app, typically represented by icons (such as the prohibition sign "⊘") or text like "Remove ads by ...".
- **Reward Element**: Elements indicating that a user would receive a reward for required actions (*e.g.*, watching video ads), represented by icons or text like "unlocking new features" or "increasing the game coins and items."
- **Voluntary Ad Trigger (VAT) Element**: Elements indicating that an ad will appear after clicking on them, often featuring icons resembling a video player's play button ("▷").
- **Asymmetric Element Pair**: A pair of elements that prominently highlight the option benefiting developers using bright colors or bold fonts.
- **Disguised Element**: An element visually imitating iOS system UIs such as notification or alert popups.

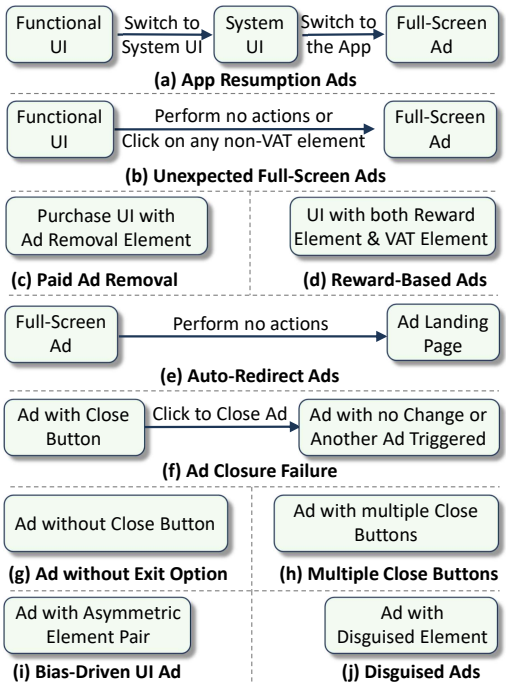


Fig. 4: UI-interaction traces of ads dark patterns

(3) User Interactions Possibly Helpful for Triggering Dark Patterns. A user interaction performed on a given UI consists of a user action and the specific target element. We identified five types of actions essential for exploring dark patterns: (i) *Click*: a tap on the UI page; (ii) *Tap and drag*: the user briefly taps the screen and then drags the finger across it; (iii) *Return to Home Screen/Control Center*: the user temporarily leaves the app to return to the Home Screen or access the Control Center; (iv) *Switch Back to App*: the user brings the app from the background back to the foreground; (v) *Idle*: the user remains on the current UI without performing any input or action.

2) Sequential Representations for Dark Pattern Identification: After scrutinizing dark pattern instances in AdsDP, we found that instances of the same type typically share a common representation: a sequential UI-interaction trace comprising UIs, UI elements, and user interactions. Detecting dark patterns is essentially identifying sequences matching this representation.

(1) Representation of Dark Patterns as Sequential UI-Interaction Traces. Fig. 4 illustrates the representation of the ten dark patterns, which is also described below. A dark pattern is essentially a chain of interactions from a starting UI to a landing UI, reflecting its real-world observable manifestation.

- **`App Resumption Ads`** involve three UIs, where the user switches from an app functional UI to a system UI and then back to the app, triggering a full-screen ad.
- **`Unexpected Full-Screen Ads`** occur across two UIs, where a full-screen ad appears after either no action or interaction with non-VAT elements on a functional UI.
- **`Paid Ad Removal`** and **`Reward-Based Ads`** are static patterns defined within a single UI. The former appears on a purchase UI with an ad removal element, while the latter involves the co-occurring reward and VAT elements.

TABLE III: Manual validation results of the proposed sequential representations across ten ads dark pattern categories.

Dark Pattern Category	# of Instances	# of Mismatched Instances
App Resumption Ads	13	0
Unexpected Full-Screen Ads	62	0
Auto-Redirect Ads	38	1
Paid Ad Removal	89	23
Reward-Based Ads	112	0
Ad Without Exit Option	255	0
Ad Closure Failure	102	0
Multiple Close Buttons	10	0
Bias-Driven UI Ads	8	0
Disguised Ads	8	0
Total	**697**	**24**

- **`Auto-Redirect Ads`** involve two UIs, where a full-screen ad automatically transitions to a landing page after a short period without user action.
- **`Ad Closure Failure`** spans two UIs, where clicking the close button fails to dismiss ad or triggers another ad.
- **`Ad Without Exit Option`** and **`Multiple Close Buttons`** are static patterns, each represented within a single UI, characterized by the absence of a close button or the presence of multiple close buttons, respectively.
- **`Bias-Driven UI Ads`** and **`Disguised Ads`** are static patterns, each represented within a single UI and characterized by an asymmetric element pair or a disguised element within an ad, respectively.

(2) Robustness Evaluation of the Proposed Representations. To evaluate the credibility and robustness of these representations, we randomly sampled 697 dark pattern instances across ten categories from AdsDP and manually checked whether each instance's behavior aligns with sequential representations. As detailed in Table III, 673 instances (96.56%) exactly matched our sequences, while the remaining 24 mismatched.

We examined the 24 mismatches, which included 23 `Paid Ad Removal` and 1 `Auto-Redirect Ads` cases. All mismatches deviated from our proposed sequences by only a single UI element or UI. For `Paid Ad Removal`, the exact match requires an ad removal element on a purchase UI, whereas in the 23 mismatch cases, the ad removal element appeared on a non-purchase UI. However, further interaction with that element consistently led to a purchase UI, which still satisfies the essential pattern of `Paid Ad Removal`. Therefore, we slightly adjusted our detection process to accommodate such cases. For the `Auto-Redirect Ads` mismatch, the full-screen ad automatically transitioned to a blank page rather than an ad landing page, causing a mismatch. We accordingly relaxed our detection criteria to include such cases.

Summary. In summary, we learned from the AdsDP video dataset how to trigger and identify dark patterns. Consequently, we identified five effective attempts to trigger dark patterns, five categories of UI pages, seven categories of UI elements, and five types of user interactions typically involved in ads dark patterns. We also proposed sequential representations for all ten dark pattern categories and demonstrated their accuracy and robustness in dark pattern identification through a manually verified experiment. Additionally, we gained rich knowledge

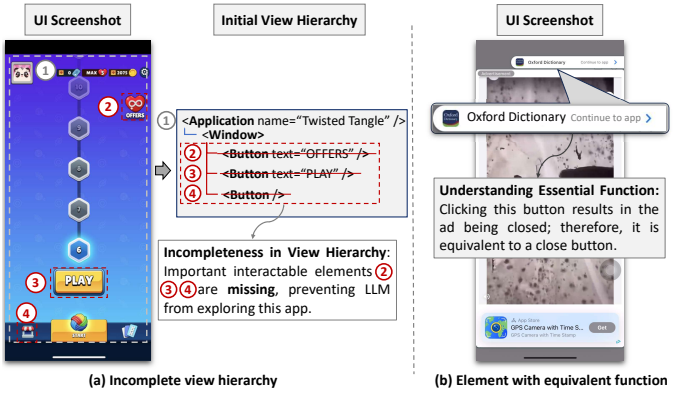


Fig. 5: Two cases illustrating challenges in UI representation and understanding

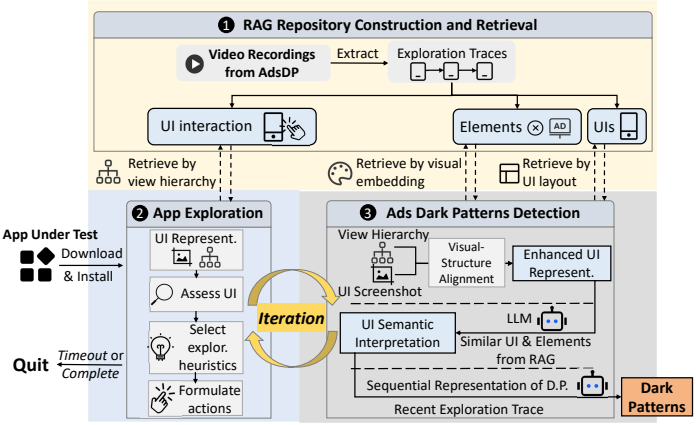


Fig. 6: The workflow of DPSCAN

about human auditors' exploration strategies (*e.g.*, which UI pages to prioritize, what actions to perform on which elements) for specific dark patterns. Such multi-step reasoning is absent from prior detection studies and existing automated tools. These insights inspire our design of an automated system for efficient exploration and detection of dark patterns in mobile ads.

IV. DPSCAN APPROACH

We examine the challenges of automated ads dark pattern detection and present DPSCAN's design and implementation.

A. Challenges of Automated Discovery of Ads Dark Patterns

Large-scale automated auditing of ads dark patterns in mobile apps requires a GUI automation system that can effectively trigger and identify potential instances. Such a system must execute target apps on mobile devices, apply domain-specific expertise to trigger dark patterns, and analyze resulting UIs to determine pattern occurrence. Despite progress in GUI testing [35], [36] and dark pattern detection [5], [11], several unique challenges remain.

- **Incomplete UI Representation.** A mobile UI is typically represented by a screenshot and a view hierarchy—the former provides visual cues, while the latter offers a structured tree-like textual representation of UI components and their metadata (*e.g.*, coordinates, bounding boxes). However, UI elements such as close buttons are often absent from the view hierarchy in apps built on game engines like Unity [13], which powers over 50% of mobile games [37]. This absence obstructs LLM-based automation. As shown in Fig. 5(a), three interactive elements (② ③ ④) visible in the left screenshot are missing from the right view hierarchy, preventing LLM exploration.
- **Difficulty in Understanding UI Elements with Equivalent Functionality but Varying Appearances.** UI elements that are functionally equivalent may appear visually different, leading to misclassification. In Fig. 5(b), the text button "`Continue to app >`" dismisses a full-screen ad, making it functionally equivalent to a close button—a nuance that LLMs struggle to grasp.
- **Difficulty in Formulating UI Navigation Strategies Under Ambiguous Task Specifications.** After understanding the current UI and its elements, an LLM must decide which interactions are most likely to trigger a dark pattern. However, the high-level goal of detecting any of the ten dark pattern categories considered in this study cannot be directly decomposed into executable subtasks. Therefore, the LLM must intelligently devise exploration strategies that infer page-specific actions toward the overall objective.

B. Overview of DPSCAN

We present DPSCAN, a knowledge-augmented, LLM-driven system for exploration and detection of dark patterns in mobile ads. It operates end-to-end: given an app, it returns a list of detected dark patterns within a dynamic test period. By referencing fine-grained human auditors' knowledge described in §III, DPSCAN addresses the challenges outlined in §IV-A, achieving comprehensive UI understanding, near-human automated navigation, and accurate dark pattern identification.

Fig. 6 illustrates DPSCAN 's workflow. Our preliminary analysis of AdsDP [12]—a video dataset documenting numerous ads dark pattern instances—yielded a large repository of annotated UIs, UI elements, and auditors' interactions involved in dark pattern occurrences. This repository is organized into three RAG databases (❶): UI, UI element, and UI interaction, providing domain-specific references via advanced retrieval.

For a given app under test, DPSCAN connects to a mobile device through an instrumentation agent to automatically download and install the app. It then iteratively executes two modules—app exploration (❷) and dark pattern detection (❸)—over a specified test window. In the exploration module, a crafted Chain-of-Thought (CoT) prompt guides the LLM to select the most promising exploration heuristic among those detailed in §III and to make action decisions by referencing human auditors' interactions from the UI-interaction RAG database as exemplars. In the detection module, a multi-step LLM-driven method emulates human reasoning: it first obtains the UI representation (screenshot and view hierarchy) and enhances it with a computer-vision technique to recover missing elements; then retrieves similar UIs and elements from

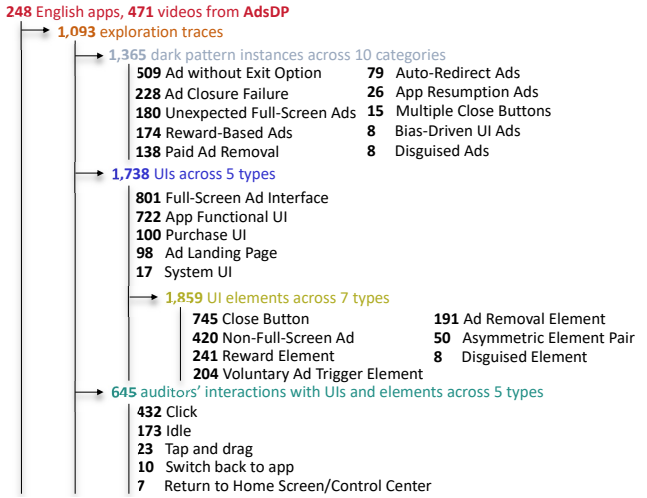


Fig. 7: Distributions of dark patterns, UIs, elements, and auditors' interactions in RAG repository

the RAG databases to enable semantic categorization of the current UI and its elements. Finally, by examining the recent exploration trace—which may contain dark pattern–related UIs, elements, and interactions—it identifies dark pattern instances matching common sequential representations. At the end of each round, the instrumentation agent executes the formulated actions, triggering UI transitions for the next round. This iterative process between the exploration and detection modules terminates when either the time limit is reached or all exploration heuristics have been sufficiently executed.

We adopt Gemini-2.5-Flash as the underlying LLM powering DPSCAN' core components, owing to its strong multi-modal reasoning and ability to execute coherent instructions under long-context inputs (including UI screenshots and textual descriptions). Notably, DPSCAN is model-agnostic and can be implemented with other LLMs offering comparable capabilities.

C. Module ❶: RAG Repository Construction and Retrieval

LLMs alone lack domain-specific knowledge of ads dark patterns, which hinders precise UI understanding and effective navigation (see §IV-A). To address this, we enhance LLMs with RAG by providing relevant exemplars. We describe the construction of RAG repository and efficient retrieval methods.

1) Repository Construction: Two researchers systematically examined 471 videos from AdsDP covering manual exploration of 248 apps, extracted clips of auditors' attempts to trigger dark patterns, sampled keyframes from them, and annotated the relevant UIs, elements, and interactions, resolving any disagreements through discussion. Inter-annotator agreement (measured by Krippendorff's α) was 0.85, indicating high annotation consistency. As shown in Fig. 7, this process identified 1,365 dark pattern instances across 988 of 1,093 exploration traces[1]. The remaining 105 traces contained no dark patterns. `Ad without Exit Option` was the most

[1]An exploration trace is a chain of interactions from a starting UI to a landing UI; a single trace may contain multiple dark patterns, as a single attempt may trigger multiple patterns.

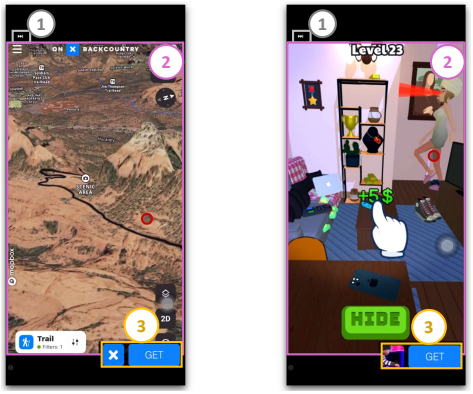


Fig. 8: Two full-screen ad interfaces sharing a similar layout

prevalent dark pattern category (509, 37.29%), followed by `Ad Closure Failure` (228, 16.70%), `Unexpected Full-Screen Ads` (180, 13.19%), `Reward-Based Ads` (174, 12.75%), and `Paid Ads Removal` (138, 10.11%). The remaining categories—`Auto-Redirect Ads` (79, 5.79%), `App Resumption Ads` (26, 1.90%), `Multiple Close Buttons` (15, 1.10%), `Bias-Driven UI Ads` (8, 0.59%), and `Disguised Ads` (8, 0.59%)—were less frequent.

For all identified dark patterns, we annotated all involved UIs, UI elements, and auditors' interactions, resulting in a repository of 1,738 UIs, 1,859 UI elements, and 645 interactions. A breakdown by category is provided in Fig. 7. The repository is organized into three RAG databases (UI, UI element, and UI interaction[2]) to supply domain-specific references.

2) Retrieval Mechanisms: We propose three retrieval mechanisms, each tailored to one of the three databases.

(1) Layout-based retrieval on the UI database. A UI contains diverse elements, making it challenging for a single encoder to compute inter-UI similarity. Layouts capture inter-element relationships and reflect designers' intent, thus are more robust than visual appearance for matching similar UIs. As in Fig. 8, two ads differ visually but share nearly identical layouts: a small button at top-left (①), a large central video area (②), and a rectangular "GET" button at bottom-right (③). Hence, the UI category (*Full-Screen Ad Interface*) annotated on the left UI can be transferred to the right UI via layout similarity.

For each UI in the database, we reconstruct its layout using UIED [38], a vision-based method that segments screenshots and extracts visible elements with coordinates and bounding boxes. For a query UI, we extract its layout (spatial coordinates and bounding boxes of elements) from the view hierarchy. Following [39], we formulate layout similarity as an optimization problem maximizing cumulative pairwise similarity over aligned element pairs, solved by dynamic programming:

(a) Sort elements. Sort elements in each layout spatially (top-to-bottom, left-to-right).

(b) Compute pairwise similarity. For any pair of elements, compute similarity based on spatial proximity (Euclidean distance of centers) and geometric consistency (ratios of

[2]A UI interaction comprises a UI and a user action performed on it.

width and height). Specifically, let two UI layouts be represented as ordered element sequences $\mathcal{U} = \{e_i\}_{i=1}^{m}$ and $\mathcal{V} = \{f_j\}_{j=1}^{n}$. Each element is associated with a bounding box $\mathbf{b} = (x_{\min}, y_{\min}, x_{\max}, y_{\max})$. The spatial similarity between two elements is defined based on the Euclidean distance between their centers:

$$s_p(e_i, f_j) = \exp\left(-\frac{\|\mathbf{c}_i - \mathbf{c}_j\|_2}{\alpha \cdot D}\right), \tag{1}$$

where $\mathbf{c}_i$ and $\mathbf{c}_j$ denote the center coordinates of e_i and f_j, respectively, D is the screen diagonal length used for normalization, and α is a decay factor, set to 0.25 as in [39]. The similarity of element shapes is defined based on their width and height ratios: $s_w(e_i, f_j) = \exp(-|\log \frac{w_i}{w_j}|)$, $s_h(e_i, f_j) = \exp(-|\log \frac{h_i}{h_j}|)$. where w and h denote the width and height of the bounding box, respectively. The final similarity between two elements is defined as: $S(i,j) = s_p(e_i, f_j) \cdot s_w(e_i, f_j) \cdot s_h(e_i, f_j)$.

(c) Maximize total similarity. Apply a weighted longest common subsequence (LCS) algorithm via dynamic programming to find the optimal alignment maximizing total similarity. Specifically, we define the dynamic programming state $\mathrm{DP}(i,j)$ as the maximum cumulative similarity between the first i elements of $\mathcal{U}$ and the first j elements of $\mathcal{V}$. The state transition is given by

$$\mathrm{DP}(i,j) = \max \begin{cases} \mathrm{DP}(i-1,j), \\ \mathrm{DP}(i,j-1), \\ \mathrm{DP}(i-1,j-1) + S(i,j), \end{cases} \tag{2}$$

where $S(i,j)$ denotes the similarity between elements e_i and f_j. The boundary conditions are defined as $\mathrm{DP}(0,j) = 0, \quad \mathrm{DP}(i,0) = 0$. And the final similarity between the two UI layouts is obtained as $\mathrm{Sim}(\mathcal{U}, \mathcal{V}) = \mathrm{DP}(m,n)$.

This method robustly identifies the most similar layout by considering element count, spatial position, and geometric shapes, with $O(mn)$ time complexity (m, n = element counts).

(2) Visual-embedding-based retrieval on the element database. For each element in the database, we use Gemini Embedding 2 [40] to convert its screenshot into a vector embedding, stored in a ChromaDB [41] vector store with an index mapping images to embeddings.

Given a query element screenshot, we retrieve similar elements considering both semantic meaning and visual design. Specifically, we compute cosine similarity between its embedding and textual descriptions of dark-pattern-related element categories to determine the best semantic match. From the primary and secondary matching categories, we retrieve the element with highest image embedding similarity, requiring a threshold of 0.6 (empirically set) to ensure basic visual commonality. We also directly retrieve the element with the highest overall embedding similarity to the query, enforcing a stricter threshold of 0.9 to ensure strong visual overlap. Results from both methods are merged. These visual overlap thresholds (0.6 and 0.9) ensure sufficient visual correspondence with query elements, allowing the LLM to treat retrieved elements as meaningful references.

(3) View-hierarchy-based retrieval on the UI interaction database. Choosing the right action on a UI with many candidates is challenging; referencing human decisions in similar UIs is crucial. Since view hierarchies structurally organize dark-pattern-related elements that auditors typically interact with, we retrieve interactions based on similarity of element distributions within the hierarchy.

Given a UI with its view hierarchy, we remove elements lacking dark-pattern-related semantics, then compute similarity against database hierarchies using unordered tree edit distance [42], efficiently calculated with APTED algorithm [43], [44].

Cache for fast startup. Retrieval features (*e.g.*, element embeddings, UI layouts) are computed once during initial run and cached locally for efficient system startup.

D. Module ❷: App Exploration for Triggering Dark Patterns

In each round of app testing, DPSCAN must decide which interactions to perform on the current UI to trigger potential ads dark patterns. To accommodate the difficulty in UI navigation (in §IV-A), we deploy an LLM-driven analysis to select the most promising exploration heuristic from those detailed in §III-A, ensuring that it is immediately executable and likely to trigger at least one ads dark pattern. The system also retrieves up to five (an empirically derived number) UI interactions to provide human interaction exemplars as grounded references for action formulation. To support the inherent multi-step reasoning required by this task, we present the Chain-of-Thought (CoT) reasoning process below:

(1) Assess the Current UI. Evaluate the current UI state by analyzing its primary functions and interactive elements (*e.g.*, buttons and input fields), with particular attention to popups that require immediate handling, such as permission requests.

(2) Recognize repetitive actions. Analyze the recent interaction history to identify repetitive actions, and figure out potential mitigation strategies accordingly (*e.g.*, restarting or clicking alternative buttons) to avoid getting stuck.

(3) Evaluate exploration progress. Evaluate the progress of each exploration heuristic (*completed/processing/failed*) based on interaction history, UI transitions, and the steps/time already consumed, then assess its feasibility on the current UI.

(4) Analyze retrieved exemplars. Sequentially analyze each reference UI interaction, compare UI states, infer the auditor's action intent and the underlying exploration heuristic, and extract transferable knowledge.

(5) Prioritize exploration heuristics. Combine the current UI state and the progress of each exploration heuristic to rank all of them based on the following principles, with the most promising one ranked first:

- **Balancing heuristic progress.** Assign low priority to exploration heuristics completed or failed, while promoting long-dormant heuristics.
- **Ensuring operational feasibility.** Prioritize a heuristic that is immediately actionable on the current UI, thereby maximizing its operational feasibility.

(6) Formulate next interactions. Design the next action according to the fixed action sequence of the selected heuristic, justify the decision, and provide human interaction exemplars from similar UIs as supporting evidence. To recover from failures (*e.g.*, app freezes or crashes), the system restarts the app when the same UI representation is observed in three consecutive interactions.

This step-by-step reasoning process emulates the multi-step process of human decision-making, guiding the LLM to evaluate the feasibility of different exploration heuristics based on feedback from past attempts and the current UI state, thereby enabling balanced progression. The prompt template used in this module is shown in Listing 1:

```
<instruction>
    Given the {action execution history}, {
    current UI (screenshot & view hierarchy)},
     {auditors' action on similar UIs}, and {
    exploration heuristics}, employ reasoning
    with following steps: (1) Assess the
    current UI, analyzing interactive elements
    . (2) Evaluate action execution history,
    recognizing repetitive actions. (3)
    Evaluate the progress of each exploration
    heuristic with one of the following
    statuses: {completed/ processing/ failed}.
     (4) Analyze each reference exemplar,
    inferring auditors' action intent. (5)
    Select a heuristic guiding your
    exploration in this step, which is
    immediately executable and does not repeat
     previously completed or failed heuristics
    . (6) Propose a sequence of actions to
    execute the selected heuristic, guided by
    exemplars.
</instruction>

<output_format>
    Output JSON object containing following
    fields: "heuristic status" (a dict listing
     every heuristic with its status), "
    selected heuristic" (string), and "
    proposed actions" (a list of JSON, each
    item containing "action type" (string) and
     "target element id" (integer)).
</output_format>

<referencing_UI-interactions>
    1. {[UI 1], [interaction 1 (interaction
    type, target element)]};
    ...
    n. {[UI n], [interaction n (interaction
    type, target element)]}.
</referencing_UI-interactions>
```

Listing 1: Prompt template for heuristic selecting and action decision-making

E. Module ❸: Ads Dark Patterns Detection

The detection procedure comprises three key components: UI representation enhancement, UI semantic interpretation, and dark pattern identification. We detail each below.

1) UI Representation Enhancement: To address the incompleteness of view hierarchies (see §IV-A), we implement a *visual-structural alignment* method that segments and extracts UI elements from screenshots and integrates missing ones into the view hierarchy. Specifically, we use UIED [38] to extract visible elements with their coordinates and bounding boxes. To avoid omissions, we apply contour detection and template matching from OpenCV [45] to identify rectangular or circular regions as potential buttons, based on our observation that most interactive elements follow such regular shapes. Only non-overlapping elements are integrated into the view hierarchy. This method ensures a complete UI representation, extending detection to apps built on engines like Unity.

2) UI Semantic Interpretation: After obtaining a complete UI representation, we retrieve similar UIs and elements from the corresponding databases and feed all information (current UI screenshots, view hierarchies, similar UIs and elements) into the LLM for fine-grained semantic interpretation. This overcomes the challenge of understanding functionally equivalent elements with varying appearances (see §IV-A).

Because view hierarchies contain many elements irrelevant to dark pattern detection, providing similar elements for all of them could distract the LLM and increase time and token costs. We therefore introduce a lightweight preliminary analysis to select only dark-pattern-related elements. The LLM rapidly scans the screenshot and view hierarchy to extract IDs of candidate elements. To mitigate potential omissions, we augment the candidate list with additional elements having typical button sizes or shapes.

The system retrieves up to three similar elements for each candidate dark-pattern-relevant element and five similar UIs for the current UI. These empirical limits balance comprehensive knowledge provision with acceptable input length. The retrieved results serve as references for the LLM, which then categorizes the current UI and its elements. This produces a structured output: a list of visible elements with their categories, and a UI profile (including overall category and interpretation). The prompt template used is shown in Listing 2.

```
<instruction>
    Given the {UI representation (screenshot,
    view hierarchy)}, {similar UIs and
    elements}, employ reasoning with the
    following steps: (1) Match elements' IDs
    in view hierarchy with visible elements in
     screenshot. (2) Analyze similarities and
    differences between the current UI and
    retrieved similar UIs, then interpret the
    current UI and select the best-matched
    semantic category for it. (3) Analyze each
     UI element that is visible and
    potentially involved in dark patterns,
    compare it with retrieved similar elements
     in appearance and functional essence,
    then categorize each element into one dark
     pattern-related category or as a non-
    related element.
</instruction>

<output_format>
```

```
    Output a JSON object containing the
    following fields: "UI categorizing" (a
    JSON object containing "category" (string)
     and "interpretation" (2 - 3 sentences,
    summarizing the UI and the rationale for
    categorization)), and "element
    categorizing" (a list of JSON, each item
    containing "category" (string) and "
    reasoning" (2 - 3 sentences)).
</output_format>

<referencing_similar_UIs>
    {[UI 1], [semantic category 1]}; ... {[UI
    n], [semantic category n]}.
</referencing_similar_UIs>
<referencing_similar_elements>
    1. Elements similar to [element screenshot
     1] in the current UI:
        {[element 1_1], [category 1_1]};
        ...
        {[element 1_m], [category 1_m]}.
    ...
    n. Elements similar to [element screenshot
     n] in the current UI:
        {[element n_1], [category n_1]};
        ...
        {[element n_m], [category n_m]}.
</referencing_similar_elements>
```

Listing 2: Prompt template for UI semantic interpretation

3) Dark Pattern Identification: Based on the categorization of UIs and elements, DPSCAN instructs the LLM to analyze the temporal trace of the last n rounds[3] and match it against the sequential representations of ads dark patterns. We empirically set $n = 3$, because the most complex pattern, `App Resumption Ads`, fully manifests within three consecutive UIs across three rounds. The analysis outputs a structured list of detected dark pattern instances, each with its constituent UIs, elements, interactions, and a detailed interpretation. To accommodate instances that deviate slightly from the predefined sequences (as observed in the manual validation of §III), we allow the LLM to detect patterns that differ by at most one UI or element from the standard sequence. This step uses the prompt template in Listing 3.

```
<instruction>
    Given the {exploration trace (UI 1 ->
    interaction 1 -> UI 2...)}, and {textual
    descriptions and sequential
    representations of dark patterns}, detect
    dark patterns by checking whether any
    subsequence in the trace matches the
    sequential representation of dark patterns
    . A mismatch of at most one UI or element
    is allowed; in such cases, perform fine-
    grained reasoning based on textual
    descriptions.
</instruction>

<output_format>
    Output JSON object containing the
    following fields: "dp list" (a list of
    JSON objects, each item containing "dp
    category" (string), "reasoning" (several
    sentences), and "evidence ui and elements"
     (integer indices of UIs and elements that
     match the sequential representation)).
</output_format>
```

Listing 3: Prompt template for dark pattern identification

[3]A round executes the exploration and detection modules once each.

V. EVALUATION

We evaluate DPSCAN by answering three research questions:

- **RQ1**: How effectively does DPSCAN explore and detect ads dark patterns in apps?
- **RQ2**: How does human expertise in DPSCAN contribute to detection performance?
- **RQ3**: How does DPSCAN compare with state-of-the-art tools for app exploration and dark pattern detection?

A. Experimental Setup

Datasets. We constructed two datasets:

- ***KNOWN***: From the 248 English apps in AdsDP, we selected 97 apps that are compatible with our environment and extensively use ads dark patterns. These apps span 20 categories, with *Games*, *Education*, and *Photos* being the most frequent. To prevent circularity and data leakage, DPSCAN's RAG repository loads only UI-interaction traces extracted from apps other than the one currently under test.
- ***UNKNOWN***: We randomly sampled 2,048 apps from the iOS App Store, varying in popularity (by rating count). They cover 23 categories, dominated by *Games* (263, 12.84%), *Utilities* (217, 10.60%), and *Education* (174, 8.50%).

Environment. DPSCAN controls an iPhone XR (iOS 14.4.1) using open-source tools: IPATool [46] for automated app downloads, Frida [47] for lifecycle management and system-level interactions (*e.g.*, switching between the home screen and apps), Appium [48] for extracting UI screenshots and view hierarchies, and ZXTouch [49] for coordinate-based in-app clicks. We selected iOS 14.4.1 for compatibility with these third-party tools and observed no noticeable impact of this older version on detection performance.

B. RQ1: Overall Performance on KNOWN Dataset

We ran DPSCAN on each *KNOWN* app for 25 minutes (∼21 rounds per app). DPSCAN recorded an exploration trace (*i.e.*, a chain of interactions from a starting UI to a landing UI) per round. Across 97 apps, it completed 2,070 rounds, performed 1,973 interactions, visited 892 unique UIs, and observed 367 ads (3.78 per app). As Table IV shows, DPSCAN detected dark patterns in 86 apps. The most frequent categories by app are `Ad Without Exit Option` (68 apps), `Ad Closure Failure` (53), and `Unexpected Full-Screen Ads` (49); the least common is `Disguised Ads` (6). In total, DPSCAN identified 1,243 dark pattern instances across 982 traces. `Ad Without Exit Option` (425 instances), `Paid Ad Removal` (207), `Reward-Based Ads` (195),

TABLE IV: Overall performance of DPSCAN on the *KNOWN* dataset

Dark Pattern Category	# Apps Detected (DPSCAN)	# Apps Detected (Manual)	# Apps Detected (Both)	Precision'(%)	Recall'(%)	# of Instances (DPSCAN)	# of Instances (Manual)	Precison(%)	Recall(%)	F1(%)
App Resumption Ads	21	22	20	95.24	90.91	27	25	92.59	100.00	96.15
Unexpected Full-Screen Ads	49	53	47	95.92	88.68	96	95	94.79	95.79	95.29
Auto-Redirect Ads	28	31	26	92.86	83.87	48	48	93.75	93.75	93.75
Paid Ad Removal	45	46	44	97.78	95.65	207	204	98.55	100.00	99.27
Reward-Based Ads	43	45	43	100.00	95.56	195	186	94.36	98.92	96.59
Ad Without Exit Option	68	73	68	100.00	93.15	425	422	95.29	95.97	95.63
Ad Closure Failure	53	58	52	98.11	89.66	148	139	93.92	100.00	96.86
Multiple Close Buttons	17	17	15	88.24	88.24	44	39	88.64	100.00	93.98
Bias-Driven UI Ads	14	12	12	85.71	100.00	41	39	90.24	94.87	92.50
Disguised Ads	6	5	5	83.33	100.00	12	12	91.67	91.67	91.67
Benign Traces	/	/	/	/	/	1,088	1,117	98.62	96.06	97.32
Total	86	85	85	**96.51**	**91.71**	2,331	2,326	**96.65**	**96.86**	**96.76**

and `Ad Closure Failure` (148) are most numerous, while `Disguised Ads` (12) the fewest.

Detection Comprehensiveness. To assess the comprehensiveness of DPSCAN's detection, two well-trained researchers established ground truth[4] by manually exploring each *KNOWN* app following the AdsDP auditing procedure (denoted *Man-Exp*) and annotating observed dark pattern categories. Manual exploration reveals dark patterns in 85 apps (87.63%), with `Ad Without Exit Option`, `Ad Closure Failure`, and `Unexpected Full-Screen Ads` being the most prevalent, appearing in 73, 58, and 53 apps, respectively.

Per-app comparison shows strong agreement between DPSCAN detection and *Man-Exp* annotations: all 85 apps identified manually were also detected by DPSCAN. For prevalent categories, both detected `Ad Without Exit Option` in 68 apps and `Ad Closure Failure` in 52 apps. Using *Precision'=# Apps Detected (Both) / # Apps Detected (*DPSCAN*)* and *Recall'= # Apps Detected (Both) / # Apps Detected (Manual)*, DPSCAN achieves average precision' of 96.51% (range 83.33–100%) and average recall' of 91.71%, indicating DPSCAN's broad coverage across all dark pattern categories.

Detection Accuracy. To evaluate DPSCAN 's detection accuracy, we also labeled each of 2,070 traces as containing dark pattern instances or not, producing 953 dark-pattern traces with 1,209 instances (422 `Ad Without Exit Option`, 204 `Paid Ad Removal`, *etc.*) and 1,117 benign traces (*i.e.*, containing no dark patterns). Compared to this ground truth, DPSCAN attains an average precision of 96.65%, recall of 96.86%, and F1 of 96.76%. Specifically, DPSCAN achieves over 90% precision for all categories except `Multiple Close Buttons` (88.64%). Notably, recall reaches 100% for `App Resumption Ads`, `Paid Ad Removal`, `Ad Closure Failure`, and `Multiple Close Buttons`.

C. RQ2: Impact of Human Expertise on DPSCAN Performance

Given the importance of human expertise, we conduct ablation studies to evaluate its contribution to UI exploration (Module ❷) and dark pattern identification (Module ❸).

[4]Although AdsDP provides existing annotations, they were collected a year ago and no longer reflect the current UI design of these apps.

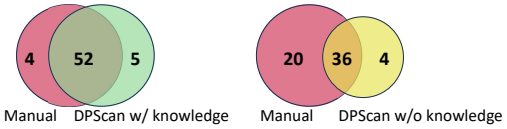


Fig. 9: Dark pattern instances identified by manual exploration (*Man-Exp*) and DPSCAN (with and without human expertise)

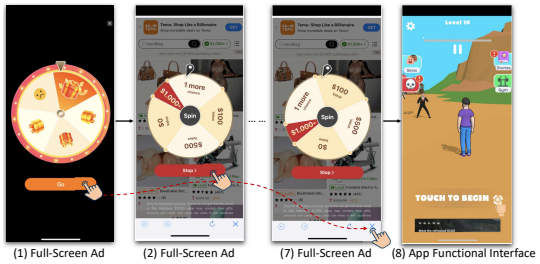


Fig. 10: A case of ineffective actions w/o exploration heuristics

1) Impact on UI Exploration: We removed human knowledge (exploration heuristics and retrieved interaction exemplars) from Module ❷, forcing the LLM to devise its own strategy. We sampled 23 apps from the *KNOWN* dataset, ran the module for 25 minutes per app without expertise, reviewed traces, manually annotated dark pattern instances, and compared against ground truth from manual exploration (*Man-Exp*).

As shown in Fig. 9, manual exploration identified 56 dark pattern instances across the 23 apps (average 2.43 per app). With human expertise, DPSCAN triggered 57 instances, 52 of which overlapped with manual exploration (Recall'=92.86%). Without expertise, performance dropped to 40 instances, with 36 overlapping (Recall' = 64.29%). A comparison of LLM decisions with and without expertise shows that human expertise helps *avoid ineffective actions* and *perform necessary actions for less common patterns*.

- **Avoiding ineffective actions.** Fig. 10 shows a case from the app *SlowMoRunner*. Without human-guided heuristics, the LLM performs repetitive, ineffective interactions (clicking "Go/Stop" on the spinning wheel) that match no sequential representation of dark patterns, leaving the ad stuck on the current page. In contrast, guided by *Attempting to Dismiss Full-Screen Ads*, it immediately clicks the "X" button in the bottom-right corner, identifies `Ad Closure Failure`, and closes the ad in only two interactions.

TABLE V: Detection module's performance under different knowledge settings

Knowledge Settings	Precision(%)	Recall(%)	F1(%)
Baseline: {Textual description of dark patterns}	84.11	87.89	85.96
{Sequential representation}	89.97(↑5.86)	89.97(↑2.08)	89.97(↑4.01)
{Sequential representation + Similar UI/elements}	**92.78(↑8.67)**	**93.43(↑5.54)**	**93.10(↑7.14)**

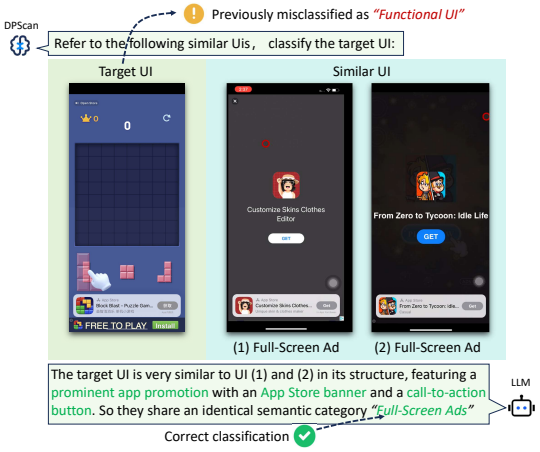


Fig. 11: A case of correcting misclassification via similar UIs

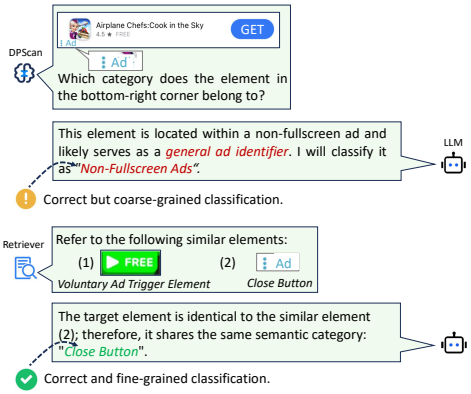


Fig. 12: A case of fine-grained LLM understanding via similar elements

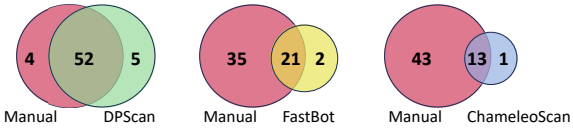


Fig. 13: Dark pattern instances identified by manual exploration (*Man-Exp*), DPSCAN, FastBot, and ChameleoScan

- **Performing necessary actions.** Without expertise, the LLM tends to overlook the *Return to Home Screen/Control Center* and *Switch Back to App* interactions, likely because these complex actions trigger only one pattern, `App Resumption Ads`. With the heuristic *Switching between an in-app UI and the smartphone home screen*, however, the LLM makes selective but necessary switches between the app and system UI, detecting `App Resumption Ads` in the app "DrawingDesklite".

2) Impact on Dark Pattern Identification: We ablated two forms of knowledge in the detection module: sequential representations and similar UI/element instances. We collected 207 auditors' exploration traces from AdsDP (26 without dark patterns), containing 263 dark pattern instances across all categories (`Ad Without Exit Option` most frequent, 90 instances). We applied Module ❸ under three settings: (1) baseline with only textual descriptions of dark patterns, (2) adding sequential representations, (3) adding both sequential representations and similar UI/element instances.

Table V summarizes the results. The baseline achieved an F1 of 85.96%. Adding sequential representations raised F1 to 89.97%, and including both yielded the best performance (F1 = 93.10%, exceeding baseline by 7.14%). Among 15 UI/element categorization errors, similar UIs/elements resolved 11 (73.33%) by *correcting misclassifications* and *refining coarse-grained understanding*.

- **Correcting misclassification.** Fig. 11 shows a UI misclassification case from *cheese.themepack*. The target, a *Full-Screen Ad Interface* with a playable game demo and only a bottom ad banner as a clue, is misread by the LLM as a normal gaming interface and labeled *App Functional UI*. Based on the UI layout, DPSCAN matches two similar *Full-Screen Ad Interfaces* sharing the same bottom banner design. Using this banner as an anchor, the LLM infers that the target UI follows the same design rationale and correctly classifies it as a *Full-Screen Ad Interface*.
- **Refining coarse-grained UI understanding.** In Fig. 12, the bottom-left icon functions to close the ad when clicked and should therefore be categorized as the ad's *Close Button*. Without similar elements for reference, the LLM relies on surrounding context (*i.e.*, the non-full-screen ad containing it) and coarsely labels it as a *Non-Full-Screen Ad*. With similar elements, however, it references prior auditor annotations to correctly classify it as a *Close Button*. This benefit is especially valuable for common ad assets about which the LLM lacks prior knowledge.

D. RQ3: Comparison with State-of-the-Art Works

We compare DPSCAN's Exploration Module (❷) against existing app exploration frameworks, and its Detection Module (❸) against prior dark pattern detection tools.

1) Comparison with Existing App Exploration Frameworks: To assess DPSCAN's exploration effectiveness, we compare it with two state-of-the-art iOS automation frameworks, ChameleoScan [35] and ByteDance's FastBot [50], on the 23 *KNOWN* apps used in the ablation study (§V-C1). Each framework is run for 25 minutes per app, and the discovered dark pattern instances are manually annotated and compared with the ground truth from manual exploration (*Man-Exp*).

As shown in Fig. 13, DPSCAN triggers 57 instances, 52 of which overlap with *Man-Exp* (Recall' = 92.86%), substantially outperforming FastBot (21 overlapping instances, Recall' = 37.50%) and ChameleoScan (13 instances, Recall' = 23.21%). A review of exploration traces reveals that triggering dark patterns hinges on visiting key UIs and elements (*e.g.*, *Purchase UI* and *Full-Screen Ad*) rather than maximizing general UI coverage. For instance, although FastBot executes 314 steps

TABLE VI: Comparison of DPSCAN against DPGuard and UIGuard on four dark pattern categories and benign UIs.

Category	Precision(%)			Recall(%)			F1(%)		
	DPScan	DPGuard	UIGuard	DPScan	DPGuard	UIGuard	DPScan	DPGuard	UIGuard
Reward-Based Ads	97.44	94.44	**100.00**	**97.44**	87.18	20.51	**97.44**	90.67	34.04
Paid Ad Removal	94.12	75.00	**100.00**	**100.00**	75.00	18.75	**96.97**	75.00	31.58
Disguised Ads	**100.00**	42.86	33.33	**87.50**	75.00	25.00	**93.33**	54.55	28.57
Bias-Driven UI Ads	**87.50**	44.44	0.00	87.50	**100.00**	0.00	**87.50**	61.54	0.00
Benign UI	**89.29**	62.07	30.49	**89.29**	64.29	89.29	**89.29**	63.16	45.45
Average	**93.94**	69.03	38.38	**93.94**	78.79	38.38	**93.94**	73.58	38.38

and visits 49 unique UIs per app on average, its low Recall' (37.50%) indicates that most explored UIs are unrelated to the task of triggering ads dark patterns.

2) Comparison with Existing Dark Pattern Detection Tools: DPGuard [5] and UIGuard [9] are state-of-the-art tools for mobile UI dark pattern detection[5]. Among these, four categories—`Paid Ad Removal`, `Reward-Based Ads`, `Bias-Driven UI Ads`, and `Disguised Ads`—are commonly supported by DPGuard, UIGuard, and DPSCAN.

Because UIGuard and DPGuard cannot explore apps autonomously and are limited to static single-screen detection, we manually extracted 99 UIs from AdsDP, covering four dark pattern categories—`Reward-Based Ads` (39), `Paid Ad Removal` (16), `Disguised Ads` (8), and `Bias-Driven UI Ads` (8)—along with 28 benign UIs (no dark patterns). We then compared these tools with DPSCAN 's *Module* ❸ *(Dark Pattern Detection)* on these UIs.

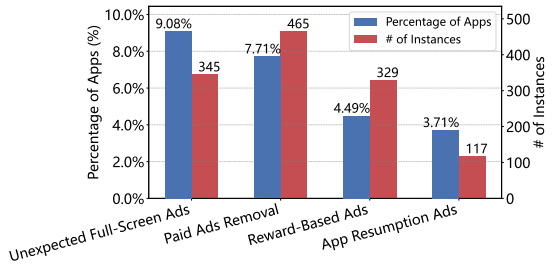
As Table VI shows, DPSCAN achieves the highest precision, recall, and F1 across most categories, with an average F1 of 93.94% (range: 87.50–97.44%), outperforming DPGuard (73.58%) and UIGuard (38.38%) by 20.36% and 55.56%, respectively. We analyzed the six misclassified instances and identified two causes: (1) four cases in `Reward-Based Ads`, `Disguised Ads`, and `Bias-Driven UI Ads` where the LLM misclassified or omitted key elements, likely due to small element sizes or limited knowledge in the LLM and RAG database; (2) two cases in `Reward-Based Ads` and `Paid Ad Removal` where the LLM did not strictly follow the sequential representation, thus missing key evidence UIs or elements (*e.g.*, labeling `Reward-Based Ads` without a *Voluntary Ad Trigger Element*).

We examine UIGuard and DPGuard's inherent limitations.

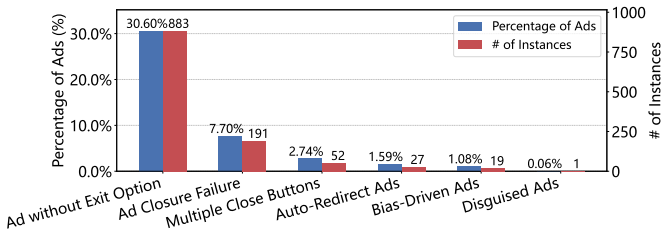
Error Analysis on UIGuard. UIGuard's rule-based detection fails to accurately model dark patterns, yielding precision and recall below 35% for most categories. For example, it defines `Disguised Ads` as "*image showing the word 'ad(s)'.*" Under the oversimplified rule, six *Ad Removal Elements* (*e.g.*, ㉮) were falsely flagged as disguised ads. Such errors highlight the weakness of rigid textual/visual representations. Conversely, DPSCAN uses semantic categorization of UIs and elements via LLM understanding, handling diverse designs.

Error Analysis on DPGuard. DPGuard relies on vague descriptions, leading to LLM misinterpretations and low precision for `Disguised Ads` (42.86%) and `Bias-Driven UI Ads` (44.44%). For instance, it detects `Bias-Driven UI Ads` using the prompt "*observe whether any option appears more significant than other equivalent choices*" without defining "equivalent." This forces the LLM into unnecessary comparisons across non-equivalent elements, causing three false positives. In contrast, our sequential representation provides clear UI/element indicators, enabling reliable detection.

[5]The work AidUI [10] shares no common categories with ours and is thus excluded from comparison.



(a) App-level dark pattern distribution on apps



(b) Ad-level dark pattern distribution on ads

Fig. 14: Ads dark pattern distribution in *UNKNOWN* apps

E. Findings on UNKNOWN Dataset

We used DPSCAN to explore 2,048 apps in the wild (25 minutes per app), quantify dark pattern prevalence, and responsibly report our findings to Apple's App Review Team. On these apps, DPSCAN performed on average 17 exploration rounds and 16 interactions per app (costing approximately 0.13 USD, 197.2K input tokens, and 47.6K output tokens per app). It identified 374 apps (18.26%) containing 1,256 app-level dark pattern instances (3.36 per app). Among 745 apps that displayed ads, the first ad appeared after an average of 5.81 interactions; 237 apps (31.81%) showed ads immediately at launch, and 550 (73.83%) within the first three rounds. In total, 1,572 ads were observed (2.11 per app). Of these, 563 ads (35.81%) employed 1,173 ad-level dark patterns (2.08 per ad).

Validation. We manually validated the detections on 258 randomly sampled traces from 100 *UNKNOWN* apps. Among these traces, 94 contain no dark pattern. On this set, DPSCAN achieved a precision of 91.04%, recall of 95.85%, and F1 of 93.38%. Moreover, we manually inspected the exploration traces of 50 *UNKNOWN* apps and found no instances of the five excluded dark patterns (*i.e.*, categories with fewer than eight instances in AdsDP), suggesting that they are uncommon and unlikely to affect the completeness of our findings.

Distribution by category. As shown in Fig. 14, app-level patterns are dominated by `Unexpected Full-Screen Ads` (9.08% of apps, 345 instances) and `Paid Ad Removal`

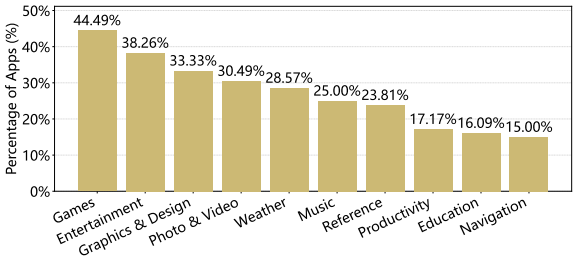


Fig. 15: Top 10 app types using dark patterns

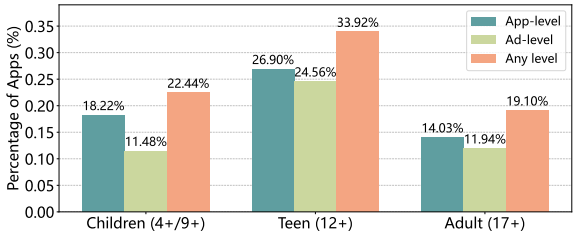


Fig. 16: Dark pattern distribution across app age ratings

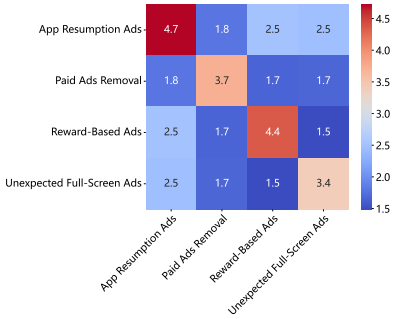
(7.71%, 465). `Reward-Based Ads` (4.49%, 329) and `App Resumption Ads` (3.71%, 117) are less common. Among ad-level patterns, `Ad without Exit Option` is most prevalent (883 instances identified in 30.60% of ads), followed by `Ad Closure Failure` (7.70%, 191), `Multiple Close Buttons` (2.74%, 52), `Auto-Redirect Ads` (1.59%, 27), `Bias-Driven UI Ads` (1.08%, 19), and `Disguised Ads` (0.06%, 1).

Prevalence by app type. Fig. 15 presents the top 10 app types (out of 23) with the highest dark pattern prevalence. *Games* (44.49%) and *Entertainment* (38.26%) exhibit the highest dark pattern adoption, while utility apps like *Education* (16.09%) and *Navigation* (15.00%) use them less, likely because their monetization relies on premium features rather than ad revenue.

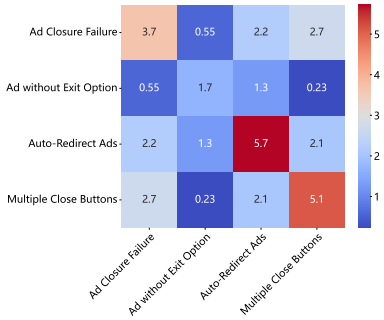
Prevalence by age rating. As shown in Fig. 16, dark patterns appear across all age groups. Apps rated for teens (12+) have the highest prevalence (33.92%), followed by children's apps (4+/9+, 22.44%), while adult-rated apps (17+) show lower but still substantial rates (19.10%). This indicates that teens and children are more exposed to dark patterns than adults, despite likely being more vulnerable and receiving additional app store protections [7], [8], [51].

Co-occurrence. Dark patterns are seldom deployed in isolation; developers may combine multiple patterns within the same app or ad to amplify their deceptive effects. To quantify this tendency, we compute the pointwise mutual information (PMI) for each pattern pair, defined as $\text{PMI}(x, y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$. A PMI score > 0 indicates positive association (*i.e.*, co-occurrence more often than by chance), whereas a score ≤ 0 indicates independence or negative correlation.

Fig. 17 presents the PMI scores for app-level (17(a)) and ad-level (17(b)) pairs as symmetric heatmaps (diagonal values indicating self-information are omitted). Among app-level patterns, `App Resumption Ads` most frequently co-occurs with `Reward-Based Ads` (PMI = 2.5) or `Unexpected Full-Screen Ads` (PMI = 2.5). At the ad level, `Ad Closure Failure` and `Multiple Close Buttons` form the most frequent pairing (PMI = 2.7).



(a) App-level dark pattern PMI heatmap



(b) Ad-level dark pattern PMI heatmap

Fig. 17: PMI heatmaps of ads dark patterns

F. Generalizability to Other Dark Pattern Categories/Instances

To demonstrate DPSCAN's generalizability beyond AdsDP, we evaluate it on three independent datasets of general UI dark patterns: RICO' [9], ContextDP [10], and AppRay [52]. We select six representative dark pattern categories: three shared with AdsDP (`Paid Ad Removal`, `Reward-Based Ads`, and `Disguised Ads`) and three additional categories: `Privacy Zuckering` (forced to share more privacy), `Nagging` (interruptive promotional pop-ups), and `Roach Motel` (easy registration but difficult account deletion).

We summarize the UI elements involved and derive corresponding UI-interaction traces for detecting the three additional dark patterns. Specifically, we introduce three UI elements: *Popup Element* (a floating window in the UI), *Privacy Policy Element* (privacy policy text), and *Toggle Element* (a binary control, *e.g.*, a checkbox). The derived sequential representations are illustrated in Fig. 18: `Privacy Zuckering` is characterized by a *Privacy Policy Element* without a *Toggle Element* that offers users a genuine choice; `Nagging` involves a *Popup Element* that displays ads, in-app purchases, or upgrade prompts; and `Roach Motel` is represented by text specifying that accounts must be deleted via email or phone. These sequential representations achieve a 100% match rate across 60 sampled instances from the three datasets. We then incorporate these traces into DPSCAN to support the three additional dark patterns and evaluate its detection module (❸) on 488 annotated instances spanning all six categories.

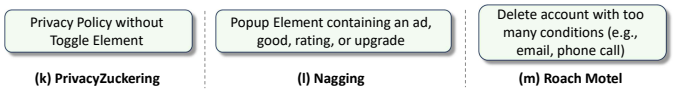


Fig. 18: UI-interaction traces of three additional dark patterns

TABLE VII: Detection module's performance on independent datasets

Dark Pattern Category	# of Instances	Precision(%)	Recall(%)	F1(%)
Paid Ad Removal	132	98.36	90.91	94.49
Reward-Based Ads	13	100.00	92.31	96.00
Disguised Ads	67	95.31	91.04	93.13
Privacy Zuckering	147	100.00	95.92	97.92
Nagging	124	96.49	88.71	92.44
Roach Motel	4	100.00	100.00	100.00
Total	488	98.03	91.99	94.92

Table VII summarizes the results. DPSCAN achieves an average precision of 98.03%, recall of 91.99%, and F1-score of 94.92%, with 100% precision for `Reward-Based Ads`, `Privacy Zuckering`, and `Roach Motel`. The slightly lower recall for `Nagging` (88.71%) stems from the LLM missing some novel *Popup Elements*, which could be mitigated by enriching DPSCAN's RAG repository with such cases.

G. Robustness to LLM Choice

To evaluate the impact of the foundational LLM, we replace Gemini-2.5-Flash with GPT-4o and Kimi-2.5 (both with multi-modal reasoning capability) across all LLM-dependent components of DPSCAN. We sample 15 apps from the *KNOWN* dataset, run DPSCAN with each LLM for 25 minutes per app, manually annotate dark pattern instances in the traces, and compare the results with the Gemini-2.5-Flash ground truth from the overall experiment (§V-B) to assess detection comprehensiveness and accuracy across LLM backbones.

TABLE VIII: DPSCAN's performance across LLM backbones

Model	Precision'(%)	Recall'(%)	Precision(%)	Recall(%)
Baseline: Gemini-2.5-Flash	**94.56**	78.57	**95.89**	**87.29**
GPT-4o	94.03(↓0.53)	70.24(↓8.33)	91.86(↓4.03)	79.66(↓7.63)
Kimi-2.5	90.66(↓3.90)	**80.95(↑2.38)**	89.02(↓6.87)	84.32(↓2.97)

Table VIII summarizes the results. DPSCAN achieves stable performance across different LLM backbones, with all metrics varying by less than 10%. Recall' shows the largest variation (70.24%–80.95%), suggesting that LLM selection mainly affects exploration capability. Among the models, Gemini-2.5-Flash consistently outperforms the others, achieving the highest Precision' (94.56%), Precision (95.89%), and Recall (87.29%), while Kimi-2.5 demonstrates potential for dark pattern exploration, achieving the highest Recall' (80.95%). These results confirm that DPSCAN is model-agnostic, with its effectiveness primarily stemming from the framework rather than any specific LLM.

H. Robustness under Adversarial Attacks

We emulated a developer attempting to evade DPSCAN by constructing adversarial traces from 40 previously recorded exploration traces from the *KNOWN* dataset. First, inspired by prior work on UI perturbation [53] and prompt injection [54], we applied visual and textual obfuscation to mislead the LLM's categorization of UIs and elements:

- *Visual prompt injection*: embedding deceptive text in screenshots to override system instructions;
- *Noise element insertion*: adding misleading close buttons or AdChoices logos ▷;
- *Textual distortion*: replacing words with obfuscated characters (*e.g.*, "video" to "v1de0");
- *Layout deconstruction*: masking canonical assets (*e.g.*, the "GET" button GET) to hinder layout-based inference.

Second, we disrupted the original traces to break our sequential representations of dark patterns by:

- *Transition interface injection*: inserting an ambiguous transition UI that is difficult to categorize semantically;
- *Static pattern splitting*: distributing a static pattern's elements across two consecutive UIs.

We provide examples of each adversarial attack in [55].

TABLE IX: Adversarial attack results

Attack Method	# Successful Bypasses	Bypass Rate(%)
Visual prompt injection	2 out of 7	28.57
Noise element insertion	0 out of 8	0.00
Textual distortion	0 out of 9	0.00
Layout deconstruction	1 out of 8	12.50
Transition interface injection	0 out of 2	0.00
Static pattern splitting	0 out of 6	0.00
Total	**3 out of 40**	**7.50**

As Table IX shows, only 3 of 40 (7.50%) adversarial traces bypassed DPSCAN. *Visual prompt injection* achieved the highest bypass rate (28.57%), followed by *layout deconstruction* (12.50%). However, such modifications—stress tests against DPSCAN —would confuse real users and degrade user experience, rendering them impractical in practice. Notably, both trace-disruption attacks (transition interface injection and static pattern splitting) failed entirely, showing that the LLM retains sufficient flexibility for nuanced judgment even when guided by sequential dark pattern representations.

VI. Discussion

Practical Implications. DPSCAN offers practical value to multiple stakeholders. For the research community, our open-source datasets provide a reusable benchmark for future studies on UI automation and dark pattern detection. For auditors such as app stores and lawmakers, DPSCAN can be deployed as both a pre-release screening tool and a post-release auditing system, enabling scalable compliance monitoring with reproducible assessments. For the public, the system provides an independent, data-driven reference that complements existing user reviews, generating transparency that empowers users to make informed choices about the apps they download and trust.

Policy Recommendations. Our large-scale analysis also yields insights for improving dark pattern governance. The high prevalence of dark patterns in teen-rated apps (33.92%) suggests that young users face elevated exposure risks. Auditors should therefore consider stricter compliance checks for teen-oriented apps and prioritize proactive auditing of high-risk categories, such as games and entertainment. More broadly, our findings

suggest a shift from complaint-driven enforcement toward proactive, scalable auditing, in which automated tools such as DPSCAN can identify policy violations before they affect users. Our findings further highlight the need for standardized and actionable auditing criteria: our UI-interaction traces provide a structured, essential representation of dark pattern manifestations, which can help platforms and regulators develop more consistent review guidelines.

Limitations. DPSCAN currently supports only iOS apps, but it employs no platform-specific designs and can be ported to Android with minimal changes to device-interaction APIs. The system also struggles with complex multi-step tasks (*e.g.*, gaming or account registration), likely due to its limited set of atomic actions (*e.g.*, click, tap & drag). While DPSCAN primarily targets ad-related dark patterns, its knowledge-augmented design—*e.g.*, sequential representations—can be readily extended to detect a broader range of dark patterns. Additionally, the system's reliance on LLM inference introduces computational overhead: each app evaluation costs approximately $0.13, which, while modest for research, may be non-negligible for large-scale continuous auditing. Finally, although DPSCAN resists most adversarial attacks (7.5% bypass rate), sophisticated adversaries with white-box access to the LLM or RAG database might design more effective evasion.

Future Work. We envision DPSCAN as a tool for app store vetting and regulatory oversight. Reducing financial and computational overhead is a priority—*e.g.*, by optimizing prompt length, caching retrieval results, or using smaller specialized models for routine decisions. Another direction is to expand the atomic action set (*e.g.*, keyboard input, gesture sequences) to handle login flows and in-app registration. To improve robustness, future versions could incorporate adversarial training or dynamic update of the RAG repository based on newly discovered dark pattern variants.

VII. Conclusion

In this study, we systematically investigate human auditors' expertise in triggering and identifying ads dark patterns, distilling heuristic exploration strategies and formalizing UI-interaction sequences for ten dark pattern categories. Based on this expertise, we construct a RAG repository of UIs, UI elements, and user interactions to provide reference exemplars for dark pattern detection. Building on these insights, we propose DPSCAN, a knowledge-augmented, LLM-driven system that automates the exploration and detection of dark patterns in mobile ads. DPSCAN implements a multi-step workflow that integrates human-derived strategies and retrieval-based exemplars throughout the exploration and detection process. Extensive evaluation on a verified dataset and 2,048 real-world apps demonstrates strong detection accuracy (96.76% F1), broad coverage (91.71% recall), and robustness against adversarial attacks. Our large-scale empirical study further reveals the prevalence and strategic co-occurrence of dark patterns across app categories and age ratings, highlighting the exposure risks for younger users. DPSCAN's open-source implementation and dataset empower regulators, researchers, and end-users to conduct effective, scalable dark pattern auditing and mitigation.

VIII. Ethical Considerations

This study was conducted in accordance with the ethical guidelines established by our institution's ethical review board (ERB). All data used for RAG repository construction and evaluation were sourced from AdsDP, a publicly available dataset. The sole exception is the *UNKNOWN* dataset, for which we referenced app ratings publicly available on the App Store to gauge app popularity. All third-party components integrated into DPSCAN are open-source libraries. During automated interactions, DPSCAN is explicitly instructed to avoid actions involving payments, purchases, or personal data submission, and it does not store or transmit user-generated content over the open internet. Finally, because the experiments in this study did not involve human subjects, no further formal ethical approval was required under our institution's policies.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No.62272410). Haitao Xu is the corresponding author.

References

[1] F. B. Insights. (2025) Mobile advertising market size. [Online]. Available: "https://www.fortunebusinessinsights.com/mobile-advertising-market-102496"

[2] Federal Trade Commission. (2022) Bringing dark patterns to light. [Online]. Available: "https://www.ftc.gov/reports/bringing-dark-patterns-light"

[3] L. Gak, S. Olojo, and N. Salehi, "The distressing ads that persist: Uncovering the harms of targeted weight-loss ads among users with histories of disordered eating," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, pp. 1–23, 2022.

[4] A. Mathur, G. Acar, M. J. Friedman, E. Lucherini, J. Mayer, M. Chetty, and A. Narayanan, "Dark patterns at scale: Findings from a crawl of 11k shopping websites," *Proceedings of the ACM on human-computer interaction*, vol. 3, no. CSCW, pp. 1–32, 2019.

[5] Z. Shi, R. Sun, J. Chen, J. Sun, M. Xue, Y. Gao, F. Liu, and X. Yuan, "50 shades of deceptive patterns: A unified taxonomy, multimodal detection, and security implications," *arXiv preprint arXiv:2501.13351*, 2025.

[6] L. Zhang-Kennedy, M. Keleher, and M. Valiquette, "Navigating the gray: Design practitioners' perceptions toward the implementation of privacy dark patterns," *Proceedings of the ACM on human-computer interaction*, vol. 8, no. CSCW1, pp. 1–26, 2024.

[7] G. Ltd. (2026) Latest ad policy. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9857753

[8] ——. (2026) Google play families policies. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9893335

[9] J. Chen, J. Sun, S. Feng, Z. Xing, Q. Lu, X. Xu, and C. Chen, "Unveiling the tricks: Automated detection of dark patterns in mobile applications," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–20.

[10] S. H. Mansur, S. Salma, D. Awofisayo, and K. Moran, "Aidui: Toward automated recognition of dark patterns in user interfaces," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1958–1970.

[11] A. Nayak, Y. Wani, S. Zhang, R. Khandelwal, and K. Fawaz, "Automatically detecting online deceptive patterns," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 96–110.

[12] Y. Shang, G. Wang, M. Ren, H. Zhang, X. Chen, H. Xu, C. Yue, S. Hao, B. Zhou, W. Ma *et al.*, "Adsdp: A video dataset for recognizing and examining dark patterns in ios in-app advertisements," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 9, no. 3, pp. 1–31, 2025.

[13] (2025) Unity engine. [Online]. Available: https://unity.com/

[14] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[15] Z. Guan, J. C. L. Li, Z. Hou, P. Zhang, D. Xu, Y. Zhao, M. Wu, J. Chen, T.-T. Nguyen, P. Xian *et al.*, "Kg-rag: Enhancing gui agent decision-making via knowledge graph-driven retrieval-augmented generation," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025, pp. 5396–5405.

[16] H. Brignull, M. Miquel, J. Rosenberg, and J. Offer, "Dark patterns-user interfaces designed to trick people," in *Proceedings of the Poster Presentation, Australian Psychological Society Congress, Sydney, NSW, Australia*, 2015, pp. 21–23.

[17] C. M. Gray, Y. Kou, B. Battles, J. Hoggatt, and A. L. Toombs, "The dark (patterns) side of ux design," in *Proceedings of the 2018 CHI conference on human factors in computing systems*, 2018, pp. 1–14.

[18] L. Di Geronimo, L. Braz, E. Fregnan, F. Palomba, and A. Bacchelli, "Ui dark patterns and where to find them: a study on mobile applications and user perception," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020, pp. 1–14.

[19] C. Gao, J. Zeng, D. Lo, X. Xia, I. King, and M. R. Lyu, "Understanding in-app advertising issues based on large scale app review analysis," *Information and Software Technology*, vol. 142, p. 106741, 2022.

[20] Y. Shang. (2025) (online demo) adsdp: A video dataset for recognizing and examining dark patterns in ios in-app advertisements. [Online]. Available: https://zenodo.org/records/15316373

[21] M. Kowalczyk, J. T. Gunawan, D. Choffnes, D. J. Dubois, W. Hartzog, and C. Wilson, "Understanding dark patterns in home iot devices," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–27.

[22] B. Schaffner, N. A. Lingareddy, and M. Chetty, "Understanding account deletion and relevant dark patterns on social media," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, pp. 1–43, 2022.

[23] C. M. Gray, S. S. Chivukula, and A. Lee, "What kind of work do" asshole designers" create? describing properties of ethical concern on reddit," in *Proceedings of the 2020 acm designing interactive systems conference*, 2020, pp. 61–73.

[24] J. Aagaard, M. E. C. Knudsen, P. Bækgaard, and K. Doherty, "A game of dark patterns: Designing healthy, highly-engaging mobile games," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–8.

[25] C. M. Gray, C. Santos, N. Bielova, M. Toth, and D. Clifford, "Dark patterns and the legal requirements of consent banners: An interaction criticism perspective," in *Proceedings of the 2021 CHI conference on human factors in computing systems*, 2021, pp. 1–18.

[26] T. T. Nguyen, M. Backes, and B. Stock, "Freely given consent? studying consent notice of third-party tracking and its violations of gdpr in android apps," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2369–2383.

[27] C. M. Bruhner, D. Hasselquist, and N. Carlsson, "Bridging the privacy gap: Enhanced user consent mechanisms on the web," in *Proc. NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb@ NDSS)*, 2023.

[28] N. Bielova, L. Litvine, A. Nguyen, M. Chammat, V. Toubiana, and E. Hary, "The effect of design patterns on (present and future) cookie consent decisions," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 2813–2830.

[29] V. Krauß, P. Saeghe, A. Boden, M. Khamis, M. McGill, J. Gugenheimer, and M. Nebeling, "What makes xr dark? examining emerging dark patterns in augmented and virtual reality through expert co-design," *ACM Transactions on Computer-Human Interaction*, vol. 31, no. 3, pp. 1–39, 2024.

[30] J. Gunawan, A. Pradeep, D. Choffnes, W. Hartzog, and C. Wilson, "A comparative study of dark patterns across web and mobile modalities," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–29, 2021.

[31] D. Kirkman, K. Vaniea, and D. W. Woods, "Darkdialogs: Automated detection of 10 dark patterns on cookie dialogs," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 847–867.

[32] David Wang, John Anthony Ribando, David Mo, and Nicholas Tung. (2019, 9) A chrome extension that protects consumers from marketing tricks when they shop online. [Online]. Available: https://devpost.com/software/insite-qfpjcd

[33] T. H. Soe, C. T. Santos, and M. Slavkovik, "Automated detection of dark patterns in cookie banners: how to do it poorly and why it is hard to do it any other way," *arXiv preprint arXiv:2204.11836*, 2022.

[34] R. Schäfer, P. M. Preuschoff, R. Niewianda, S. Hahn, K. Fiedler, and J. Borchers, "Don't detect, just correct: Can llms defuse deceptive patterns directly?" in *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–11.

[35] H. Lin, Y. Hu, H. Xu, Y. Lu, M. Ren, S. Hao, C. Yue, Z. Li, F. Zhang, and Y. Jiang, "CHAMELEOSCAN: Demystifying and Detecting iOS Chameleon Apps via LLM-Powered UI Exploration," in *Proceedings of the 33rd Network and Distributed System Security Symposium (NDSS 2026)*, 2026.

[36] J. Li, J. Liu, J. Mao, J. Zeng, and Z. Liang, "Ui-ctx: Understanding ui behaviors with code contexts for mobile applications." in *NDSS*, 2025.

[37] S. Team. (2025) What percentage of games are made with game engines? [Online]. Available: https://splicedonline.com/what-percentage-of-games-are-made-with-game-engines/

[38] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, "Uied: a hybrid tool for gui element detection," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1655–1659.

[39] R. Liu, X. Teoh, Y. Lin, G. Chen, R. Ren, D. Poshyvanyk, and J. S. Dong, "Guipilot: A consistency-based mobile gui testing approach for detecting application-specific bugs," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 753–776, 2025.

[40] G. Ltc. (2026) Gemini embedding 2. [Online]. Available: "https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-embedding-2"

[41] Chroma Ltd. (2026) chroma. [Online]. Available: "https://github.com/chroma-core/chroma"

[42] K.-C. Tai, "The tree-to-tree correction problem," *Journal of the ACM (JACM)*, vol. 26, no. 3, pp. 422–433, 1979.

[43] M. Pawlik and N. Augsten, "Efficient computation of the tree edit distance," *ACM Transactions on Database Systems (TODS)*, vol. 40, no. 1, pp. 1–40, 2015.

[44] ——, "Tree edit distance: Robust and memory-efficient," *Information Systems*, vol. 56, pp. 157–173, 2016.

[45] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[46] majd. (2026) Ipatool. [Online]. Available: https://github.com/majd/ipatool

[47] (2026) Frida: A world-class dynamic instrumentation toolkit. [Online]. Available: https://frida.re/

[48] (2026) Appium: Cross-platform automation framework. [Online]. Available: https://github.com/appium/appium

[49] (2026) Zxtouch: ios automation framework. [Online]. Available: https://github.com/xuan32546/IOS13-SimulateTouch

[50] Z. Lv, C. Peng, Z. Zhang, T. Su, K. Liu, and P. Yang, "Fastbot2: Reusable automated model-based gui testing for android enhanced by reinforcement learning," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022)*, 2022.

[51] K. Renaud, C. Sengul, K. Coopamootoo, B. Clift, J. Taylor, M. Springett, and B. Morrison, ""we're not that gullible!" revealing dark pattern mental models of 11-12-year-old scottish children," *ACM Transactions on Computer-Human Interaction*, vol. 31, no. 3, pp. 1–41, 2024.

[52] J. Chen, Z. Wang, J. Sun, Z. Xing, Q. Lu, Q. Huang, X. Xu, and L. Zhu, "From exploration to revelation: App-level context-aware deceptive pattern detection for mobile applications," *ACM Transactions on Software Engineering and Methodology*, 2026.

[53] C. Chen, X. Song, Y. Chai, Y. Yao, H. Zhao, L. Li, J. Li, Y. Teng, G. Liu, and Y. Wang, "Ghostei-bench: Do mobile agents resilience to environmental injection in dynamic on-device environments?" *arXiv preprint arXiv:2510.20333*, 2025.

[54] L. Wang, Z. Ying, T. Zhang, S. Liang, S. Hu, M. Zhang, A. Liu, and X. Liu, "Manipulating multimodal agents via cross-modal prompt injection," in *Proceedings of the 33rd ACM International Conference on Multimedia*, 2025, pp. 10 955–10 964.

[55] Y. Shang. (2026) Supplementary material, code and data. [Online]. Available: "https://github.com/syx1031/DPScan"