Mobile App Analysis in the New Era: Challenges and the Road Ahead

JIALE WU*, Huazhong University of Science and Technology, China
TIANMING LIU*, Huazhong University of Science and Technology, China
YANJIE ZHAO, Huazhong University of Science and Technology, China
HAOYU WANG†, Huazhong University of Science and Technology, China

Mobile applications (apps) have grown into complex, multi-platform systems that support diverse services and operate within varying regulatory and distribution contexts. Traditional analysis techniques have established a strong foundation for ensuring quality, security, and compliance, but face increasing challenges as mobile apps evolve. This paper presents a three-layer perspective to examine these challenges. The Artifact Layer highlights how heterogeneous code bases and protection mechanisms reduce the analyzability of packaged apps. The Runtime Layer addresses growing execution complexity, from multi-modal interactions and super-app architectures to AI-driven autonomous operations. The Ecosystem Layer considers broader contexts including emerging platforms, fragmented distribution channels, and region-specific regulatory requirements. New characteristics across these layers interact and compound, creating difficulties that existing approaches are not designed to handle. For each layer, we identify key challenges and potential research directions, including framework-aware analysis, adaptive testing, and policy-aware compliance verification. We also discuss how large language models bring new analytical capabilities to the field while their integration into apps raises new security and privacy concerns. This paper aims to clarify where established methods fall short and highlight directions for future research in this evolving field.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Software and application security**; • **Software and its engineering** → **Software verification and validation**; **Software reverse engineering**.

Additional Key Words and Phrases: Mobile App Analysis, Program Analysis, Mobile Security, Mobile Software Engineering, Large Language Models, Privacy Compliance

ACM Reference Format:
Jiale Wu, Tianming Liu, Yanjie Zhao, and Haoyu Wang. 2025. Mobile App Analysis in the New Era: Challenges and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (September 2025), 35 pages. https://doi.org/xx.xxxx/xxxxxxx

*Jiale Wu and Tianming Liu are the co-first authors.
†Haoyu Wang is the corresponding author (haoyuwang@hust.edu.cn).

Authors' Contact Information: Jiale Wu, jialewu@hust.edu.cn, Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China; Tianming Liu, tmliu@hust.edu.cn, Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China; Yanjie Zhao, yanjie_zhao@hust.edu.cn, Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China; Haoyu Wang, haoyuwang@hust.edu.cn, Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China.

ACM 1557-7392/2025/9-ART1
https://doi.org/xx.xxxx/xxxxxxx

1 Introduction

The rapid evolution of mobile technology has fundamentally transformed how people interact with digital devices, making mobile apps common in daily life. These apps now range from simple utilities to complex, platform-integrated systems that handle sensitive data, financial transactions, and communications. This requires methods that can ensure quality, security, and performance at scale, establishing mobile app analysis as an important field of research and development. In this paper, we use the term mobile app analysis to refer to the systematic examination of mobile apps to evaluate their implementation and behavior through automated and semi-automated techniques. Established approaches include static analysis, which inspects code implementations within app packages to recover structure and potential problems (e.g., Flowdroid [50]); dynamic analysis, which observes runtime behavior to assess actual functionality and resource usage (e.g., Sapienz [165]); and hybrid approaches, which combine static and dynamic techniques for more comprehensive evaluation. These techniques have established a strong foundation and are widely used in both academic research and industrial practice (as introduced in §2.1).

However, contemporary mobile apps are increasingly reshaped by rapid technological advances, evolving development paradigms, and changing platform constraints. Modern development practices have introduced new architectural patterns (e.g., cross-platform frameworks such as React Native [26, 70] and Flutter [12]) and execution paradigms (e.g., super-app ecosystems [184, 199] and agent-driven interactions [204, 235]) that create novel forms of complexity. These changes alter not only what needs to be analyzed but also how analysis can be effectively performed, leaving traditional approaches with blind spots. As a result, gaps have emerged between the capabilities of current analysis methods and the realities of modern apps [50, 165, 173].

To address these challenges systematically, we introduce a layered perspective that organizes emerging features of mobile apps and their analysis challenges by scope: the **Artifact Layer** (the packaged app and its code and resources), the **Runtime Layer** (the app's behavior and interactions during execution), and the **Ecosystem Layer** (the surrounding platform, distribution, and governance environment). By moving from app artifacts outward to ecosystem constraints, this framework clarifies how different types of complexity interact and compound across layers.

At each layer, we examine emerging changes, the challenges they create, and potential research directions. Taken together, our analysis indicates that the evolution of mobile apps is driven not by isolated changes, but by changes across artifacts, runtime, and ecosystems that interact and reinforce one another. Heterogeneous code bundles, more complex execution behaviors, and fragmented deployment contexts combine to produce forms of complexity that existing techniques struggle to address. The layered framework highlights key challenges: reduced analyzability in packaged artifacts, increased runtime complexity from super-app architectures, multi-modal interactions, and agent-driven operations, and greater difficulty in maintaining consistency across diverse platforms and regulatory contexts. Based on these observations, we outline research directions that combine platform-specific adaptation with broader unifying methods, aiming to sustain the effectiveness of mobile app analysis as the ecosystem continues to evolve.

The remainder of this paper is structured as follows. Section 2 reviews current mobile app analysis methods and introduces the three-layer perspective. Section 3–5 then analyze the Artifact, Runtime, and Ecosystem Layers in detail, each structured around emerging characteristics, analysis challenges, and possible research directions. Section 6 brings together the findings across layers, compares academic and industrial viewpoints, and highlights trends and insights in the evolution of mobile app analysis. Finally, Section 7 concludes the paper.

2 Mobile App Analysis: Current Methods and a Layered Perspective

Mobile app analysis has made significant progress in specific areas, but as apps evolve, they become more powerful yet increasingly difficult to analyze thoroughly and effectively. This section first reviews current mobile app analysis techniques (§2.1), then introduces our three-layer perspective to observe the landscape of mobile apps and organize the challenges (§2.2).

2.1 Overview of Mobile App Analysis

This section provides an overview of static and dynamic methods, hybrid approaches, and AI-integrated solutions that constitute the main technical foundations of mobile app analysis. These techniques have been developed over many years and are now widely adopted in both academic research and industrial practice. They form the established toolkit that researchers and practitioners rely on for examining mobile apps.

Static Analysis Techniques. Static analysis operates by examining app implementations in bytecode or binary form without execution, typically using frameworks such as Soot [127] and WALA [32]. It provides comprehensive coverage by reasoning about all possible control and data-flow paths, and has supported a wide range of security and reliability tasks. Representative usages include the detection of private data leaks [50, 66, 103, 132, 133, 160, 178, 214, 238], identification of vulnerabilities [53, 73, 84, 161, 197, 214], enforcement of permission checks [51, 80, 227], detection of malware [94, 101, 129, 130, 163, 174, 186, 196, 224, 237], and identification of app clones [72, 141]. A persistent challenge is the increasing heterogeneity of code bases: modern apps often combine managed code (Dalvik/ART bytecode) with native binaries, requiring cross-language analysis techniques [150, 185, 203] to bridge semantic gaps. While static analysis offers breadth and completeness, it often generates false positives and incurs high computational cost, especially on large or heavily obfuscated apps.

Dynamic Analysis Techniques. Dynamic analysis seeks to establish actual behavior by executing apps in controlled environments and observing their interactions with users, system services, and external resources. Representative frameworks such as Monkey [105], Sapienz [165], Stoat [189], Droidbot [138], and QTesting [173] illustrate this trajectory, with LLM-driven approaches [155, 198] representing recent advances. Tools for simulating user interactions [44, 77, 88, 139, 146, 177, 182, 209] have further improved coverage [41], GUI exploration [167, 241], and fault detection [107]. Beyond testing, dynamic analysis underpins security and privacy assessments: runtime evidence such as network traffic [52, 134, 153, 188, 211, 216], system calls [45, 56, 82], and memory snapshots [54, 229, 239] enables compliance evaluation [46, 85, 147, 171, 175, 180] and malicious behavior detection [38, 45, 49, 56, 59, 62, 63, 95, 99, 131, 159, 222, 239]. It is particularly valuable for capturing runtime-only phenomena such as dynamically loaded code [38] or unauthorized data exfiltration [192], though limited by incomplete coverage, nondeterministic behavior, and the difficulty of reproducing complex execution environments.

Hybrid Analysis Techniques. Recognizing the complementary strengths and weaknesses of static and dynamic methods, hybrid approaches integrate the two. A common pattern is to use static analysis to flag potential data-flow paths and then validate them dynamically [39, 229], or conversely to use dynamic traces to refine or confirm static signatures. Hybrid techniques have been especially influential in malware detection [45, 63, 95] and taint analysis [38, 62, 99], where neither static nor dynamic alone is sufficient. The main challenge lies in designing effective integration strategies that balance coverage, precision, and scalability.

AI-integrated Analysis Solutions. Recent years have seen the rapid infusion of artificial intelligence into mobile app analysis, leveraging traditional machine learning (ML), deep learning (DL), and increasingly, large language models (LLMs). ML-based methods have long been established in

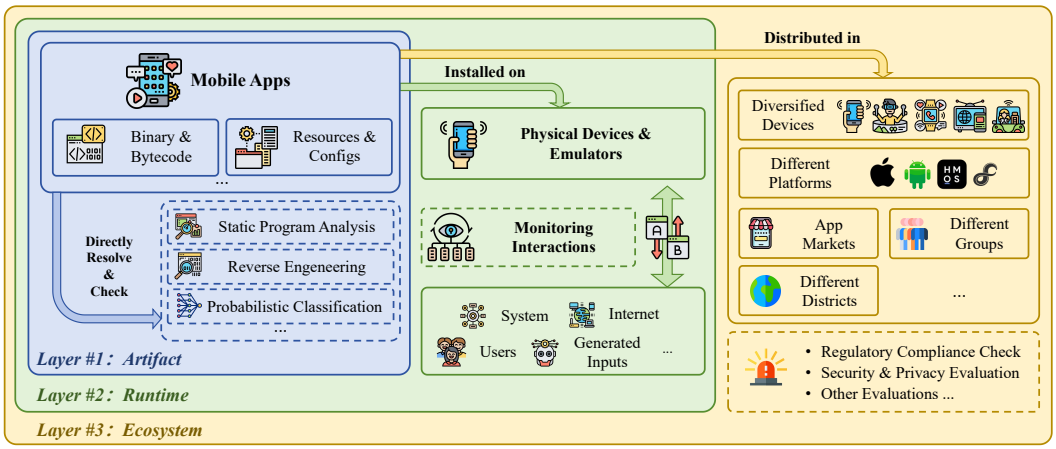

Fig. 1. Three-Layer Perspective for Examining Mobile App Analysis Challenges

malware classification [48, 166] and code similarity detection. For instance, statistical features are used to identify third-party libraries [164] and analyze malware lineage [206]. To enhance automation, DL and reinforcement learning (RL) have been introduced to guide input generation [173] and learn user interaction patterns [139] for testing. More recently, LLMs have extended these capabilities by leveraging semantic understanding: they have been applied to reproduce crashes from bug reports [116], translate natural language into GUI operations [96], and assist in reverse engineering tasks such as deobfuscation [60]. While surveys on LLM for software engineering [93] highlight their potential, effectively combining the reasoning power of LLMs with the precision of static and dynamic analysis remains a critical challenge to ensure robustness and explainability. In subsequent sections, we will explore the impact of LLM on mobile app analysis, with a focused discussion on this topic in §6.

2.2 The Three-Layer Perspective

Mobile app analysis covers diverse techniques as introduced above, while the mobile ecosystem has evolved across multiple dimensions. To systematically examine how these changes affect analysis capabilities, we present a three-layer perspective that organizes them by scope—starting from the app package itself and progressively moving outward to its surrounding environment. Figure 1 illustrates this organization. At the Artifact Layer, we focus on packaged apps as distributed bundles—the code, resources, and structures that analysis tools need to first process. At the Runtime Layer, we examine apps during execution, observing how they behave and interact in operational contexts. At the Ecosystem Layer, we consider the broader environment where apps are distributed and regulated, including diverse platforms, markets, and deployment frameworks.

This organization allows us to systematically relate changes in mobile apps to their effects on analysis capabilities. By examining each layer in turn, we can identify where traditional approaches encounter difficulties and where new capabilities are needed.

- **Artifact Layer — *Changes in packaged apps***: This innermost layer considers the app as a packaged artifact, focusing on how its internal composition influences what can be recovered and examined. Mobile app evolution has introduced heterogeneous components packaged in a bundle and increasingly sophisticated protection mechanisms, which obscure or transform implementation logic and thereby reduce transparency. These shifts challenge

the ability of analysis techniques (static or otherwise) to reason about code structure, detect data leaks [50, 214], and assess security properties.

- **Runtime Layer — *Changes during execution***: Moving outward from the packaged artifact, this layer focuses on the app in action: how it behaves when executed in real environments and how it interacts with users, system services, and external resources. Modern apps exhibit increasingly complex runtime behaviors that diverge from their static representations, including adaptive functionality, dynamic content loading, and sophisticated interaction patterns. These characteristics affect both monitoring and testing, requiring new strategies that go beyond traditional frameworks [155, 165, 173].
- **Ecosystem Layer — *Changes in platform context***: Beyond the app itself and its runtime, analysis is expected to also account for the broader ecosystem in which apps are developed, distributed, and regulated. This outermost layer defines the boundary for what analysis should target and verify, covering platform diversification, distribution fragmentation, and regulatory divergence. Such ecosystem-level factors shape compliance objectives and evidence standards, constraining and guiding analysis at the Artifact and Runtime Layers. By situating these external forces as the outer scope, this layer completes the inside-out narrative and highlights opportunities that arise when technical analysis is integrated with contextual awareness.

This layered perspective enables us to systematically analyze mobile app challenges by moving from technical implementation details to broader ecosystem considerations.

3 Artifact Layer

Our analysis starts with the app as a packaged artifact, focusing on the code in the distributed bundle and what can be inferred from it. Packaging and build practices shape the app's observable surface, deciding what remains visible from development and how logic is split across representations. We then outline the key characteristics of packaged artifacts, the challenges they pose, and possible ways to address them.

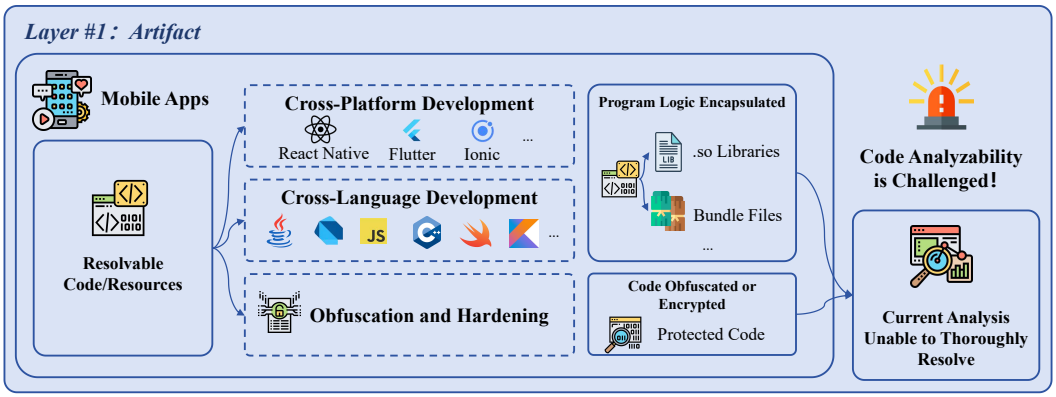

Fig. 2. Overview of Artifact Layer

3.1 Emerging Characteristics

At the Artifact Layer, the representation of implementation logic in packaged apps has shifted from uniform, analyzable bytecode to heterogeneous, framework-shaped, and protection-hardened artifacts (as illustrated in Figure 2). This section highlights the main trends that redefine what can be observed from packaged apps.

Mixed Code Architectures. Contemporary mobile apps rarely remain within a single language or runtime. Instead, they integrate managed code (e.g., Java/Kotlin bytecode) with precompiled native libraries and other components that contain implementation logic. Two specific directions dominate this trend:

- **Cross-Platform Development.** The adoption of cross-platform frameworks (e.g., React Native [26], Flutter [12], Ionic [19], and Kotlin Multiplatform [20]) has changed mobile development by enabling single-codebase deployment across platforms. A recent empirical analysis highlights their prevalence [70]: React Native appears in 14 of the top 100 apps on Google Play, while Flutter has 39 implementations in China's Tencent App Store. These frameworks differ significantly in how they package app logic: Kotlin Multiplatform leverages Java interoperability [21] to produce analyzable Dalvik bytecode, while Flutter compiles Dart code [11] into native binaries (libapp.so), and React Native bundles JavaScript code executed through bridge mechanisms [26].
- **Multi-Language Integration.** Even within a single platform, apps increasingly combine multiple languages. Android apps mix Java/Kotlin with C/C++ through the NDK, iOS apps integrate Swift with Objective-C, and many apps embed JavaScript or Lua for dynamic features. Domain-specific languages further specify resources and UI behavior. App logic thus spans Dalvik/ART bytecode, native modules, and embedded scripts within single packages.

Obfuscation and Protection. Packaging and protection mechanisms have advanced to deliberately reduce artifact transparency. Despite being a recognized challenge for over a decade, obfuscation adoption continues to grow. Common techniques include identifier renaming, string encryption and control-flow transformations [87, 207]. More advanced approaches use virtualization-based obfuscation, compiling code into custom bytecode executed by generated interpreters [125, 245]. Packing services encrypt binaries and insert loader stubs with anti-analysis logic [90], while runtime defenses such as anti-debugging, emulator detection, and integrity checks are widely deployed [55, 190]. Commercial protection services including 360 Jiagu [1], Tencent Legu [30], Alibaba MSA [24], Appdome [3], and Promon [25] offer comprehensive hardening solutions. Empirical studies show obfuscation usage in Google Play increased by 13% over the last decade [172], and sophistication continues to evolve [91, 124] despite ongoing deobfuscation research [57, 176].

Together, cross-platform development, multi-language composition, and protection mechanisms reduce the analyzability of packaged apps and increase analysis difficulty.

3.2 Analysis Challenges

The changes described above create fundamental challenges for mobile app analysis at the artifact level. The core problem is loss of analyzability: implementation logic that was previously accessible to direct inspection has become difficult to parse and understand. Traditional static analysis techniques, designed for parsing bytecode into intermediate representations and performing analysis on them [32, 50, 127], now face complexity that manifests in two interconnected ways.

Semantic Opacity and Abstraction Barriers. Cross-platform frameworks and obfuscation techniques combine to prevent clear analysis of packaged apps. Frameworks introduce abstraction layers that embed logic in non-standard representations: React Native uses JavaScript bundles executed via bridges [26], Flutter compiles Dart to native binaries [12], each requiring different analysis approaches than traditional bytecode inspection. These framework-specific transformations create barriers that conventional decompilation struggles to penetrate. Tools designed for bytecode have difficulty interpreting these components. Obfuscation compounds these barriers by deliberately obscuring whatever structure remains accessible. Virtualization-based techniques [125, 245] wrap

code in custom interpreters, control-flow transformations [87] obscure program logic, and multi-layer protection schemes combine multiple mechanisms to create artifacts that appear syntactically valid but remain semantically opaque. The result is that automated tools often fail to recover the program semantics needed for compliance verification. The additional abstraction layers create fundamental gaps in what can be analyzed.

Heterogeneous Composition. Modern apps present fragmented analysis surfaces spanning multiple languages, runtimes, and protection schemes within single packages. Existing analysis techniques have been designed to handle specific representations effectively. Static analysis frameworks are designed for examining bytecode [32, 127], native code analyzers can dissect native binaries [185, 203]. However, no single technique provides comprehensive coverage across heterogeneous compositions. In the foreseeable future, methods for analyzing various components of app artifacts are expected to emerge. It is a challenge to integrate these methods to form a comprehensive methodology.

3.3 Potential Research Directions

The challenges outlined above point toward research directions that could improve analytical capabilities for modern mobile apps. These directions address two fundamental problems: recovering semantics from obscured artifacts and handling heterogeneous code compositions.

Recovering Semantics from Framework Abstractions and Obfuscation. Cross-platform frameworks and protection mechanisms both reduce analyzability, but through different means. Frameworks introduce abstraction layers that embed code in non-standard representations: JavaScript bundles in React Native, Dart-compiled binaries in Flutter, and bridge mechanisms connecting framework code to native implementations. Analyzing these requires framework-aware techniques that understand how each framework compiles, packages, and executes code [11, 12, 26]. Existing analysis work for native binaries or React Native bundles [150, 185, 203] provides foundations, but framework-specific abstractions require specialized handling beyond general multi-language support. Protection mechanisms deliberately obscure code through obfuscation and encryption. Addressing this requires approaches that can adapt to evolving strategies rather than targeting specific techniques. AI-assisted methods show promise here: neural networks have been applied to defeat virtualization-based obfuscation [244], while LLM-based approaches demonstrate capabilities in code deobfuscation [57, 69], reverse engineering [115, 193], symbol recovery [220], and control flow reconstruction [176]. These techniques could create analysis tools that evolve alongside protection mechanisms [104, 169], though their robustness and reliability require further validation.

Unified Analysis Across Heterogeneous Compositions. Modern apps combine Java/Kotlin bytecode, C/C++ native libraries, JavaScript or Dart modules, and domain-specific resource languages within single bundles, fragmenting the analysis surface. Existing efforts such as lifting native code into Jimple IR [150, 185] or summarizing native functions for bytecode analyzers [203] show that cross-language reasoning is possible, but current solutions remain limited to narrow integration scenarios. Industrial practice suggests that broader unification is achievable. Ant Group's YASA Security Station [36], for example, integrates analyses across binaries, network logs, and host/cloud modules, orchestrating heterogeneous tools while aligning outputs into unified insights. Though designed for enterprise security rather than mobile apps, it demonstrates the feasibility of modular integration at scale. Inspired by this, we argue that mobile app analysis should pursue unified approaches that go beyond pairwise adapters. Promising directions include translating heterogeneous representations into a shared intermediate form, or developing new multi-language IRs that retain component-specific semantics while supporting whole-app reasoning. Equally important are modular architectures that can plug in specialized analyzers while tracking dependencies and

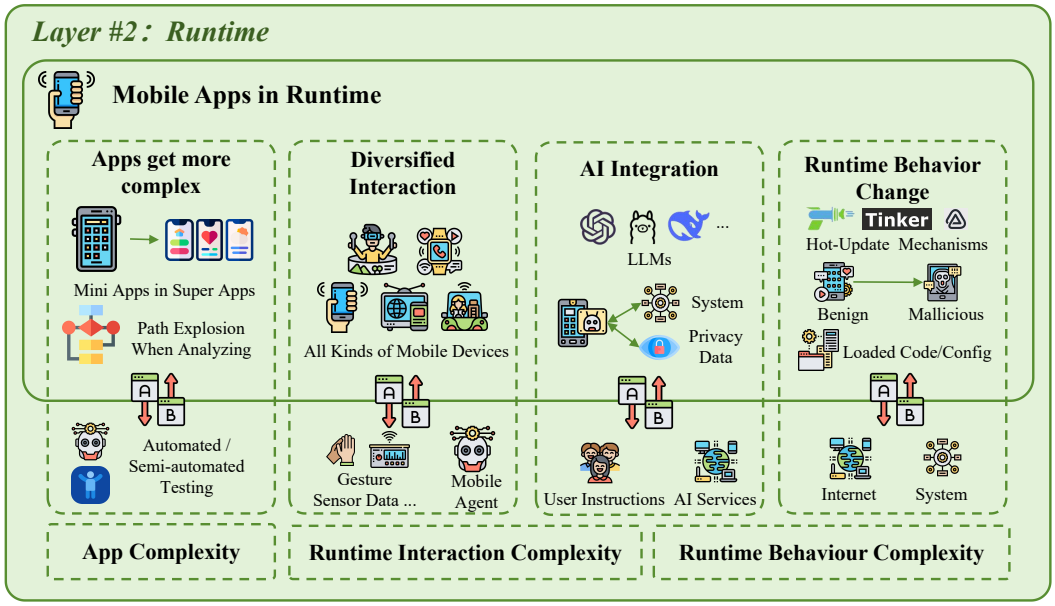

Fig. 3. Overview of Runtime Layer

control/data flows consistently across runtimes. Such approaches would provide a principled way to analyze the increasingly diverse compositions found in modern app bundles.

4 Runtime Layer

The Runtime Layer shifts the focus from packaged artifacts to the behavior of apps during execution. At this layer, we examine how apps operate in runtime environments, interact with users and system services, and adapt their functionality. Modern mobile apps exhibit increasingly complex runtime characteristics that diverge from their static representations, challenging previous analysis approaches that were designed for more predictable execution patterns.

4.1 Emerging Characteristics

At the Runtime Layer, mobile apps exhibit three interconnected dimensions of complexity that reshape how analysis is expected to be conducted (as shown in Figure 3). These characteristics reflect the evolution from simple, predictable execution patterns to dynamic, adaptive behaviors that challenge traditional analysis assumptions.

App Complexity. Modern mobile apps have evolved from simple, single-purpose tools into complex, multifunctional platforms. Many apps now integrate diverse services (messaging, payments, media streaming, social features) within single packages. The emergence of super-apps represents the extreme of this trend: platforms like WeChat combine messaging, payments, e-commerce, social networking, and mini-app ecosystems into unified apps [184]. These super-apps host millions of embedded mini-apps available for installation, with distinct functionalities, permission models, and interaction patterns [199, 228]. Many super-apps support instant installation of mini-apps within the host environment, allowing users to access services without separate downloads. The architectural complexity covers cross-component data flows, nested execution contexts, and inter-app communication mechanisms that create dense, interconnected state spaces [135, 210].

Runtime Interaction Complexity. Beyond architectural complexity, modern mobile apps operate through diverse interaction paradigms that extend far beyond traditional touch-based interfaces, covering both the diversification of interaction modalities and the emergence of autonomous agents as new interaction subjects.

- **Multi-Modal Interaction Paradigms.** Contemporary mobile interfaces support diverse interaction patterns beyond traditional touch input: gesture recognition, voice commands, augmented and virtual reality overlays, and cross-device coordination. These interaction modes are particularly prominent in emerging platform categories (AR/VR headsets, smartwatches, and automotive systems) where traditional touch-based interfaces are supplemented or replaced by spatial gestures, voice control, and contextual awareness. AR/VR interactions involve continuous spatial and temporal data streams, gesture recognition depends on timing and contextual factors, and cross-device scenarios involve coordination across multiple platforms simultaneously.
- **Agent-Driven Interaction.** Mobile apps are increasingly operated by LLM-driven agents [148, 204, 205, 215, 235] that act as intermediaries between users and apps. These agents accept natural language instructions from users, parse graphical interfaces to understand app states, and execute multi-step operations autonomously across apps and system services. Recent surveys highlight the growing importance of such computer-use agents [183], while industry reports frame agent-driven interaction as an emerging mainstream channel for mobile scenarios [168]. The agent architecture combines semantic understanding through large language models, visual interpretation of GUI elements, and system-level operation execution. This introduces a dual nature: agents can serve as testing tools that automate complex interaction sequences, but their human-like operation patterns also expose them to risks. Traditional threats targeting human users (such as phishing interfaces or deceptive GUI elements) may affect agents due to their designed mimicry of human behavior. Additionally, agents face novel attack vectors specific to LLM systems, including prompt injection and environmental manipulation [218].

Runtime Behavior Complexity. Complementing architectural and interaction complexity, app behavior is no longer static after installation. Apps now adapt their functionality at runtime through dynamic loading, hot-update, and AI integration, introducing dynamic and unpredictable behaviors that complicate analysis.

- **Runtime App Behavior Change.** Modern apps can alter their behavior after deployment through hot-update frameworks such as CodePush [9] for React Native, Shorebird [27], and Flutter Eval [13]. These enable over-the-air code delivery, bypassing app store review cycles. The mechanisms use hybrid execution engines to reconcile interpreted hotfix code with precompiled binaries, but this dual abstraction often introduces temporal inconsistencies and unpredictable state transitions. Although major distribution platforms restrict such practices [15, 28], commercial solutions including Tencent Tinker [195], Alibaba AndFix [43], Hansel [16], and Apptimize [6] remain widely used. Dynamic code injection makes post-deployment behavior potentially different from pre-release analysis, undermining stability assumptions. Recent studies document that some market-listed apps exhibit discrepancies between storefront descriptions and after-installation behavior [92, 112, 128, 246]. These "chameleon apps" have been characterized in prior research [246], showing how runtime changes can enable deceptive practices and complicate static analysis and regulatory oversight.
- **AI Integration in Apps.** The direct embedding of generative AI capabilities represents a significant trend. A recent study [111] analyzed 149 open-source Android apps incorporating LLMs, showing that these models are primarily utilized as chatbots but also perform

diverse functions such as specialized content generation, voice interaction, and task/productivity management. Integration approaches vary: some apps rely on third-party APIs from providers like OpenAI and Google, whereas others implement self-deployed APIs or embed lightweight models directly within the apps [111]. This integration paradigm enables sophisticated features including natural language interaction, intelligent agents, and autonomous decision-making components [68, 113], delivering substantial efficiency and functionality. Contemporary mobile operating systems are increasingly incorporating LLMs as autonomous agents for automated tasks [119, 219]. Also, LLM middleware frameworks (e.g., LangChain [22]) have also streamlined app creation by allowing developers to interface with LLMs through backend APIs, accelerating the growth of multifunctional apps. At the same time, this trend brings concrete risks: reliance on external APIs raises concerns over data leakage, on-device models demand heavy resources and careful isolation, and the nondeterministic nature of LLM outputs complicates testing and quality assurance.

These three dimensions of runtime complexity collectively transform the landscape for mobile app analysis. Unlike the relatively static challenges of the Artifact Layer, runtime complexity introduces temporal and contextual dependencies that make analysis inherently more difficult. The state space explosion from super-apps, the simulation challenges of multi-modal and agent-driven interactions, and the unpredictability introduced by hot-update and AI integration create a fundamentally different analytical context than what traditional dynamic analysis frameworks were designed to handle.

4.2 Analysis Challenges

The runtime characteristics outlined above create substantial challenges for mobile app analysis. These challenges affect different aspects of analysis but share a common theme: traditional approaches designed for simpler, more predictable apps struggle to handle modern runtime complexity.

Testing Complexity Challenges. The first group of challenges concerns difficulties in testing modern apps comprehensively.

- **Coverage and Scalability.** Modern mobile apps generate state spaces that are difficult to explore exhaustively through conventional testing. The combinatorial explosion of execution paths across multiple subsystems, embedded apps, and cross-component data flows makes systematic coverage increasingly difficult [135, 210]. Even recent advances in LLM-guided testing [198] and automated event sequence generation [75] face difficulties as apps grow into multifunctional ecosystems. The result is that significant portions of app functionality remain unexplored, creating blind spots in security assessment.
- **Interaction Simulation.** Multi-modal interactions involving gesture recognition, voice commands, AR/VR overlays, and cross-device coordination resist simulation in testing environments. These patterns are particularly problematic for emerging platform categories (AR/VR headsets, smartwatches, automotive systems) where apps rely on platform-specific sensors and interaction modes. These interaction patterns depend on continuous spatial and temporal data streams, nondeterministic timing, and contextual factors that are difficult to model and reproduce. Traditional GUI testing frameworks, designed for deterministic touch-based event sequences, have difficulty simulating the high-dimensional and context-sensitive behaviors required for comprehensive coverage. Testing environments struggle to accurately simulate gesture timing, spatial AR interactions, or cross-device synchronization patterns that depend on real-world contexts and hardware configurations.

Autonomous Behavior Security Challenges. The second group addresses security and privacy risks from intelligent, autonomous operations.

- **Agent-Driven Security.** As agents become mainstream interaction channels for mobile scenarios [168], they expose risks already observed in non-mobile domains that become more acute in mobile contexts. Prompt injection and manipulation attacks can steer agents toward adversarial behaviors [151, 226], backdoor and action hijacking threats can cause benign-looking tasks to trigger unintended privileged operations [240], and environmental injection attacks exploit crafted interface elements or contextual cues to influence agent decision-making [123, 140, 221]. In mobile settings, these issues become more pressing because mobile ecosystems combine complex runtime states, permission systems, and cross-app deep link mechanisms that adversaries can exploit. Crafting deceptive GUIs, poisoned deep links, or hidden triggers can induce agents into granting dangerous permissions, conducting unauthorized purchases, or exfiltrating sensitive data. Recent studies confirm that mobile LLM agents can be exploited in practice [218], meaning mobile app analysis needs to evaluate not only how apps might deceive human users but also how they can manipulate automated agents.
- **AI Integration Risks.** Embedding AI capabilities directly into apps creates privacy and security implications distinct from agent-driven risks. The diverse integration approaches (from third-party API dependencies to local model deployment [111]) create varied attack vectors spanning prompt injection [151], model manipulation, and data exfiltration. The nondeterministic nature of LLM behaviors complicates reproducible testing and verification, as the same inputs may produce different outputs. Processing sensitive user data through AI components raises compliance challenges under regulations, particularly when AI decision-making processes are opaque. Traditional analysis methods struggle to capture semantic vulnerabilities in natural language interfaces or assess privacy implications when data flows through nondeterministic AI processing, creating gaps in both security assessment and regulatory compliance verification.

Temporal Behavior Detection. Apps can change their behavior after deployment through hot-update and dynamic loading. Hot-update mechanisms enable over-the-air code delivery that can alter app functionality, making pre-release analysis insufficient for understanding actual behavior. Although officially restricted by major platforms [15, 28], commercial solutions remain adopted secretly. This enables "chameleon apps" that exhibit behavior discrepancies between storefront descriptions and installed functionality [246]. The temporal inconsistency undermines static analysis validity and creates challenges for detecting covert behavioral changes that occur after deployment. Current analysis frameworks generally lack the continuous monitoring capabilities needed to track post-deployment modifications.

These challenges show that runtime complexity has altered the requirements for mobile app analysis across multiple dimensions, from testing scale to interaction simulation to security assessment of autonomous behaviors to temporal behavior tracking.

4.3 Potential Research Directions

The challenges outlined above point toward research directions that address different aspects of runtime complexity.

Adaptive Testing Frameworks. The coverage and scalability challenges posed by super-apps and multifunctional apps demand testing approaches that move beyond exhaustive exploration toward intelligent, targeted analysis. Research should develop adaptive frameworks that use machine learning and large language models to guide state space exploration efficiently. These frameworks could incorporate several key capabilities: memory mechanisms that track exploration history to avoid redundant testing of equivalent states, learned models that predict which execution paths are more likely to show faults or security issues, and dynamic adjustment of testing strategies based

on observed app behavior. Recent work on LLM-guided testing [137, 198] and exploration with memory [236] provides foundations, but substantial work remains to handle the complexity of interconnected feature sets in modern apps. For multi-modal interactions, research should develop specialized simulation engines that can model and generate diverse interaction sequences, including gesture patterns with realistic timing variations, voice commands with different acoustic properties, and AR/VR spatial interactions that account for environmental contexts. The goal is semantic-aware testing that understands app functionality rather than merely exercising code paths.

Security Analysis for Autonomous Operations. The next two directions address security risks from intelligent behaviors, though they target different subjects and attack surfaces.

- **Agent Security Analysis.** Research should develop methods to systematically evaluate how apps might exploit or manipulate autonomous agents. This requires analyzing app interfaces for elements that could mislead agents through deceptive visual layouts, ambiguous labels, or hidden state indicators; detecting patterns that might trigger unintended agent actions through crafted deep links, permission requests, or interaction sequences; and verifying that apps do not include adversarial prompts or contextual cues designed to manipulate agent decision-making through prompt injection or environmental manipulation [140, 218, 221]. Beyond defensive analysis, research should explore using agents as analysis tools themselves. Recent surveys on GUI agents with foundation models [143, 208] suggest opportunities for agents to assist in automated testing, security assessment, and compliance verification. The challenge is developing bidirectional approaches where agents serve as both subjects requiring protection and tools providing analytical capabilities.
- **AI Behavior Analysis.** Apps that integrate AI capabilities directly require methods to characterize the space of possible AI behaviors, bounding the range of outputs an AI component might produce for given inputs, identifying conditions under which AI behavior might become problematic, and detecting when AI responses might leak sensitive information or violate privacy expectations. This includes analyzing prompt injection vulnerabilities in mobile-specific contexts where AI components interact with system services and user data [151], tracking data flows through AI components to verify compliance with privacy regulations, and developing testing approaches that can provide meaningful coverage despite nondeterminism. The diverse integration patterns observed in practice [111], such as third-party APIs, local models, and middleware frameworks, require analysis techniques that can handle varied architectures while providing consistent security and privacy assessments.

Comprehensive Runtime Behavior Monitoring. The temporal gaps created by hot-update and dynamic code loading demand continuous monitoring techniques to detect post-deployment modifications. A central difficulty is the lack of systematic detection methods: existing approaches often rely on indirect indicators such as developer metadata or market descriptions [246], but fail to reliably identify embedded frameworks or dynamic loading behaviors, leaving app marketplaces and analysis tools exposed to unmonitored changes. This problem is compounded by the divergence between installation and runtime behavior, where hot-update mechanisms allow apps to pass market review in one state but later alter functionality through OTA updates, enabling so-called "chameleon apps" that bypass pre-installation checks and pose severe security and regulatory challenges [92]. Addressing these issues requires monitoring approaches that integrate static indicators (e.g., framework signatures, bytecode patterns) with dynamic runtime evidence, ideally within sandboxed environments that preserve execution integrity, so that covert hot-update mechanisms can be detected and the risks of unvetted code injection can be reduced.

These research directions address different aspects of runtime complexity: intelligent exploration for testing scale, security analysis for autonomous operations, and temporal monitoring for

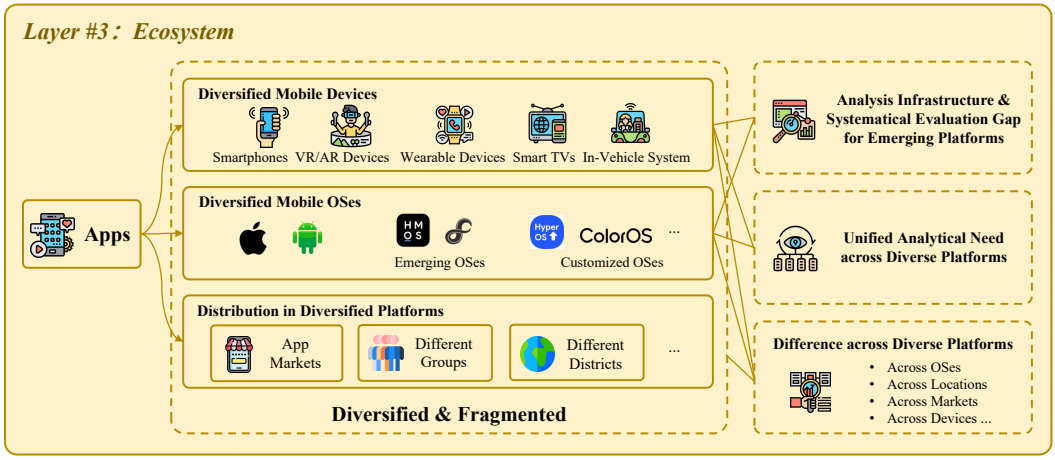

Fig. 4. Overview of Ecosystem Layer

behavioral changes. Progress requires moving beyond traditional assumptions about predictable, static app behavior toward approaches that can handle the dynamic, adaptive, and intelligent characteristics of modern mobile apps.

5 Ecosystem Layer

The Ecosystem Layer shifts the focus outward from apps themselves to the broader environments in which they are developed, distributed, and regulated. Unlike the Artifact and Runtime Layers, which examine internal structures and execution behaviors, this layer addresses how contextual diversity and distribution fragmentation reshape analysis requirements. Mobile apps now operate across an expanding range of devices, operating systems, and distribution channels. As illustrated in Figure 4, these trends transform mobile app analysis from a uniform task into a context-dependent challenge that varies across platforms, markets, and regulatory contexts.

5.1 Emerging Characteristics

From the ecosystem perspective, mobile apps are no longer confined to a single dominant platform or device type. Instead, the landscape is both *diversified*, with apps extending to new platforms, devices, and operating systems, and *fragmented*, with the same app exhibiting differences across distribution channels, user groups, and geographic regions or other dimensions. This section highlights three key aspects of this evolution: diversified devices, diversified operating systems, and diversified distribution.

Platform Diversification. Mobile platforms have diversified in two complementary ways: through the proliferation of new device categories, and the rise of emerging operating systems.

- **Diversified Mobile Devices.** Mobile platforms have extended beyond smartphones to AR/VR glasses, wearable devices, smart TVs, in-vehicle systems, and IoT hybrids, permeating daily life through apps that span health monitoring, spatial computing, smart home control, and immersive entertainment. These apps mirror smartphone functionalities while retaining platform-specific traits. Some apps operate on modified mobile operating systems: Meta Horizon OS (powering the Meta Quest series [23]) and Android Automotive OS [2] leverage Android variants, while visionOS [4] builds on iPadOS foundations. This architectural inheritance highlights opportunities to transfer mobile app analysis techniques to related fields.

- **Operating System Diversification.** Mobile apps traditionally targeted two dominant platforms, Android and iOS, but the operating system landscape has become more diverse. New mobile operating systems are emerging as viable alternatives: HarmonyOS [17] by Huawei reached approximately 19% market share in China by end of 2024 and surpassed 5% globally in early 2025, with over 1 billion active devices reported by Huawei [181]. Other systems have pursued different approaches but with limited impact: Tizen OS [31] and Ubuntu Touch [33] saw modest mobile adoption and are no longer active in the smartphone market, while Google's Fuchsia [14] was deployed on smart-home devices but has since seen reduced maintenance. Android-compatible variants such as OPPO's ColorOS [10] and Xiaomi's HyperOS [35] modify core Android functionalities while maintaining app compatibility, but these modifications can affect how apps behave [89]. Recent research shows that custom ROMs sometimes grant native privileges to pre-installed apps in ways that differ from stock Android [200], altering the security assumptions that analysis tools rely on. The proliferation of both new operating systems and customized variants creates a landscape where apps exhibit platform-specific behaviors that require corresponding analysis adaptations.

Distribution and Contextual Fragmentation. Beyond platform diversity, mobile apps exhibit fragmentation across their deployment contexts, creating variations in how the same app operates in different settings. Apps are distributed through multiple channels like official app stores like Google Play and Apple App Store, regional stores like Tencent App Store [29], Huawei AppGallery [18] and even via Telegram [110] or other underground distribution platforms. Each distribution channel applies different review processes, update schedules, and content requirements. Geographic markets introduce additional variation: apps adapt their behavior based on deployment region due to regulatory requirements, infrastructure differences, and local market conditions. Recent studies document significant geodifferences in mobile apps [126] and substantial regional variations in security and privacy features [109]. Apps also modify their interfaces, available features, and data handling based on user characteristics including age, language preferences, and accessibility needs. Age verification mechanisms vary significantly across regions and contexts [230], while privacy practices adapt to different user populations [170, 249]. This fragmentation means apps can exhibit different code and behaviors depending on their specific deployment context, making uniform analysis insufficient for understanding actual app behavior across markets and user groups.

These trends transform mobile app analysis from a uniform task into a context-dependent challenge that account for platform-specific constraints, regulatory requirements, and market variations while maintaining analytical consistency across heterogeneous environments.

5.2 Analysis Challenges

Platform diversification and distribution fragmentation introduce challenges that manifest in different parts of the analysis process. We summarize them into four categories.

Emerging Platform Capability Gaps. New operating systems and vendor-customized ROMs often lack mature infrastructures for analysis. Custom ROMs illustrate this issue: while they preserve app compatibility, they also modify core system behaviors in ways that undermine existing assumptions. Recent studies show that some ROMs grant elevated privileges to pre-installed apps, breaking Android's permission model [200]. HarmonyOS poses a broader challenge: its ArkTS language [7] and distinct execution model invalidate Dalvik/ART-based tools, creating an analytical void. Although initial efforts such as Ark-Analyzer [71] provide partial support, comprehensive static and dynamic pipelines are still missing. Dynamic testing frameworks also struggle, as instrumentation tools need to be redesigned for each new system variant. Beyond individual platforms, systematic app-level analysis remains sparse: while some work has addressed

VR apps [108] and Android TV apps [152], most research remains at the system or device level [121, 149, 179], leaving app-centric risks underexplored.

Policy and Compliance Challenges. Diverse platforms impose heterogeneous policies that complicate verification. Android's Car App Quality guidelines [8] limit video, gaming, and browsing functionality in driving mode, while Wear OS standards [34] constrain smartwatch UIs by forbidding certain elements. These rules are highly context-sensitive and not easily captured by existing analysis workflows. Moreover, compliance requirements extend beyond platform rules to jurisdiction-specific regulations such as GDPR [100], requiring app behavior to be validated simultaneously against multiple, potentially conflicting regimes.

Technical Model Diversity Challenges. Underlying execution differences add another layer of complexity. HarmonyOS replaces Android's runtime with ArkTS, introducing new abstractions. Custom ROMs alter privilege granting and permission handling. Wearables and AR/VR platforms integrate sensor-rich, context-dependent interfaces that differ fundamentally from smartphones. Current analysis tools, which assume a stable execution model and permission system, lack the flexibility to reason consistently across these heterogeneous environments.

Cross-Context Systematic Comparison Challenges. Distribution fragmentation leads to variations in app behavior across markets, stores, and user groups. Large-scale studies show significant geodifferences in functionality [126] and substantial regional disparities in privacy and security [109]. Age verification practices differ across jurisdictions and can often be circumvented via VPNs or OAuth misuse [230]. Privacy compliance analysis at scale [249] and generational differences in privacy attitudes [170] further demonstrate how contextual factors shape app behavior. Current frameworks lack the ability to systematically compare across such variations, making it difficult to ensure consistent compliance and detect region-specific risks.

These challenges show that ecosystem-level complexity requires analysis approaches that can handle infrastructure diversity, policy heterogeneity, and contextual fragmentation while providing systematic coverage across the expanding mobile landscape.

5.3 Potential Research Directions

The challenges posed by ecosystem diversification and fragmentation point toward research directions that address complementary aspects of the problem. Some directions focus on building analytical capabilities for emerging platforms, while others aim to provide systematic ways of ensuring consistency across heterogeneous environments.

Building Platform-Specific Capabilities. Emerging operating systems and customized variants require dedicated infrastructures to close current capability gaps.

- **Platform-Specific Infrastructure Development.** Develop static and dynamic pipelines that natively support new languages and runtimes, such as ArkTS for HarmonyOS [7], beyond Dalvik/ART-based approaches. This includes instrumentation frameworks and automated testing tailored to new execution semantics. While early tools like Ark-Analyzer [71] provide a starting point, more complete toolchains approaching the maturity of Android and iOS ecosystems are still needed.
- **Legacy Technique Adaptation.** Established methods such as taint analysis and GUI testing should be systematically adapted. Proven approaches remain valuable if adjusted to platform-specific permission models and interaction paradigms, helping to bridge mature knowledge with emerging needs.

Policy-Aware Verification. The heterogeneous compliance landscape across platforms and devices suggests the need for automated mechanisms that can account for diverse policy requirements. One promising direction is to develop policy-as-code frameworks that encode platform-specific

rules, such as Android Car App Quality standards [8] or Wear OS App Quality guidelines [34], as verifiable constraints integrated into analysis workflows. At the same time, these systems could be extended to incorporate jurisdiction-specific regulations (e.g., GDPR [100]), moving toward automated assessments of app behavior across fragmented deployments against both platform-level guidelines and regional laws.

Toward Unified Cross-Platform Analysis. The proliferation of diverse platforms calls for analytical approaches that can ensure consistent coverage while remaining adaptable to platform-specific constraints. On one hand, apps on emerging platforms such as wearables, AR/VR, and automotive systems expose unique context-sensitive risks, requiring scenario-specific simulation in dynamic testing environments (as discussed in §4). On the other hand, many of these apps share implementation logic through cross-platform development frameworks, which encapsulate common code bases across Android, iOS, and other platforms (see Figure 2). This shared nature provides opportunities for unified analysis that leverages commonalities rather than treating each platform in isolation. Yet current practice remains fragmented: existing tools still target platforms separately, and well-established frameworks for joint analysis are lacking. Bridging this gap requires methods that can both exploit shared implementations for efficiency and incorporate platform-specific adaptations for completeness. Developing such unified approaches represents a central challenge for mobile app analysis in the multi-platform ecosystem.

Ecosystem-Scale Comparative Analysis. The fragmentation of app distribution across stores, regions, and demographic contexts highlights the need for systematic frameworks that can detect and analyze behavioral variations across different deployment scenarios. Automated comparison systems could help identify discrepancies in features, configurations, and practices across markets, languages, and user groups, building on recent studies of geodifferences [126] and regional security variations [109]. Incorporating compliance checks with legal regimes such as GDPR [100] and enabling longitudinal tracking of app evolution across platforms and jurisdictions would further strengthen the analytical toolkit.

Together, these research directions combine platform-specific adaptation with broader unification and systematic comparison. Progress in this area will likely depend on balancing efforts to address immediate capability gaps for new platforms with the development of generalizable principles that maintain analytical relevance across an increasingly diverse mobile ecosystem.

6 Discussion

The preceding sections have examined challenges at the Artifact, Runtime, and Ecosystem Layers. To bring together observations that were relatively scattered across these sections, we focus on two specific topics here. First, §6.1 provides a more in-depth discussion of how large language models, currently a prominent technological trend, affect mobile app analysis. Then, §6.2 discusses the overall development trend of Mobile App Analysis from a multi-dimensional perspective (analytical capabilities, analytical needs, and stakeholders), aiming to provide some potentially valuable insights.

6.1 LLMs: Opportunities and Challenges for Mobile App Analysis

Among all the transformations discussed in this paper, large language models (LLMs) and related techniques (particularly intelligent agents powered by LLMs), stand out as qualitatively new factors for mobile app analysis. LLMs and agents not only enable new analytical capabilities but also introduce novel security challenges. In this subsection, we discuss how LLMs and agents are reshaping the analysis pipeline, what new capabilities and constraints they introduce, and what promising directions emerge for advancing mobile app analysis.

6.1.1 LLM as Enabler.

LLM-Assisted Dynamic Analysis. Dynamic analysis has emerged as a primary domain where LLMs are being integrated into mobile app analysis. In these approaches, LLMs serve as intelligent decision-makers that can replace human judgment in testing workflows, bringing semantic understanding to tasks that traditionally required manual expertise.

LLMs are used to guide exploration of app state spaces by making context-aware decisions about which actions to execute and which states to explore. Comprehensive testing approaches like GPTDroid [155] and LLMDroid [198] employ LLMs for functionality-aware navigation throughout testing sessions, aiming to achieve higher testing coverage. Intent-driven and scenario-driven testing methods [58, 232, 242] leverage LLMs to understand testing objectives and guide exploration accordingly. Advanced approaches incorporate memory mechanisms [236] to maintain context across testing sessions, enabling more sophisticated behavioral analysis that adapts based on observed app behavior.

Beyond navigation, LLMs are employed to generate and adapt testing inputs by understanding the semantic requirements of app interfaces. Event sequence generation [75] uses LLMs to create realistic user interaction patterns through behavioral modeling. Test migration approaches [58, 64, 242] employ LLMs to adapt test scripts across functionally similar apps by recognizing semantic equivalences between different implementations. Automated test input generation [78, 154, 156] addresses the challenge of generating valid contextual inputs by using LLMs' semantic understanding of UI components and their expected input formats. Bug reproduction tools [96, 116, 201] leverage LLMs to interpret crash reports and generate reproduction steps, replacing manual debugging analysis.

LLMs are also applied to assess app behavior quality by interpreting runtime observations against expected functionality. Non-crash functional bug detection [120, 157] demonstrates how LLMs can identify subtle malfunctions that require human-level understanding of expected app behavior, addressing cases where traditional test oracles are insufficient. Accessibility and usability issue extraction during testing [117] further illustrates how LLMs perform quality assessment tasks that traditionally required manual inspection.

These applications demonstrate how LLMs bring human-like reasoning to automated testing workflows. By supporting test navigation, input generation, and behavior evaluation, LLMs are enabling more intelligent and adaptive analysis that can handle modern app complexity. As these techniques mature, we anticipate further improvements in test coverage, bug detection accuracy, and the ability to handle increasingly complex app behaviors.

LLM-Assisted Static Analysis. While non-mobile domains have extensively adopted LLMs for static analysis, including reverse engineering [115, 193], vulnerability detection [142], and even program analysis assistance such as call graph construction [233], mobile-specific applications remain limited but show promising research potential.

Current mobile static analysis research demonstrates several emerging directions. Bytecode-level summarization and dataflow graph generation [122] uses LLMs to bridge the semantic gap between low-level bytecode and high-level behavioral understanding. Code summarization for specific purposes includes malicious behavior detection [136, 243] and secret leakage analysis [42], where LLMs interpret code semantics to identify security-relevant patterns. LLM-assisted deobfuscation [244] represents another promising area where LLMs help recover semantic information from protected applications. These methods demonstrate the potential of LLMs for mobile-specific program analysis, but their reliability and interpretability in security-critical settings remain open questions that warrant further investigation.

LLM-Driven Direct Analysis of App-Associated Data. Beyond assisting traditional static and dynamic analysis, an emerging direction leverages LLMs to directly analyze app code and associated artifacts, such as privacy policies, app descriptions, user reviews, and documentation, without relying on static analysis frameworks or runtime execution. By harnessing their semantic understanding

capabilities, LLMs can process these diverse data sources in ways that traditional analysis tools cannot, opening complementary perspectives on app behavior that go beyond code-level inspection.

LLMs are now used to analyze app-associated textual data including privacy policies, app descriptions, and user-generated content that traditional tools cannot thoroughly understand and summarize. Privacy policy analysis [79] and sensitive behavior description synthesis [158] extract privacy implications from natural language documents to verify policy-behavior consistency and compliance. These approaches enable automated assessment of what apps claim to do versus what they actually implement.

LLMs can also enable ecosystem-level studies when applied across large app corpora or extensive user-generated content. Large-scale analysis of user reviews [40] extracts privacy-related concerns and trends from thousands of apps, synthesizing user perspectives that would be impractical to analyze manually. Android documentation analysis [225] processes platform specifications to extract privacy policy-related information at scale. Regional app variant comparison [109] uses LLMs to summarize functional differences across geodistributed versions of the same apps. The distinguishing characteristic of these studies is their scale and heterogeneity: LLMs synthesize insights from diverse data sources to identify patterns, inconsistencies, and emerging trends across the app ecosystem that would remain invisible in single-app analysis.

LLMs also facilitate the translation between program behavior and natural language, making technical analysis results accessible for compliance verification and transparency purposes. Behavior-policy consistency checking [234] employs LLMs to generate natural language behavioral descriptions from static and dynamic analysis results for verification against stated policies. AI capability identification [145] and sensitive data usage intention analysis [83] demonstrate how LLMs interpret complex app behaviors for specific analytical objectives, converting technical findings into human-understandable explanations suitable for regulatory compliance and user transparency.

As LLM capabilities mature, we expect this direct analysis approach to play an increasingly important role in app compliance verification and ecosystem-level understanding, complementing traditional code-based analysis methods.

LLM-Powered Agent for Autonomous Analysis. Beyond utilizing LLM capabilities within traditional analysis workflows, LLM-powered intelligent agents represent an emerging direction for mobile app analysis. Agents possess human-like reasoning capabilities and, empowered by mechanisms like function tool integration, memory management, and iterative planning, can accomplish complex automated tasks that simple LLM invocations cannot achieve [144]. Software engineering research begins exploring agent capabilities for code analysis tasks. Agents have shown promise in vulnerability detection [231, 247, 248], reverse engineering [102, 118] and other fields highly related to software analysis. For mobile app analysis, agents' autonomous capabilities hold potential for extending beyond LLM-assisted methods. In dynamic testing, rather than invoking LLMs to guide individual test actions within fixed frameworks, agents can autonomously orchestrate testing campaigns, deciding when to switch strategies, adjusting plans based on intermediate results, and coordinating multiple analysis phases. Multi-agent frameworks [97, 212, 213] demonstrate how specialized agents can decompose complex testing tasks and coordinate their execution, handling scenarios that require adaptive synchronization across multiple components. Approaches that leverage retrieval mechanisms [98] further suggest how agents could efficiently reuse analysis experiences across similar apps, reducing redundant effort while maintaining effectiveness.

For static analysis and reverse engineering, agents' ability to actively select code segments for examination, annotate findings, and maintain analysis context through memory mechanisms offers new possibilities for handling mobile apps' inherent complexity. Recent work in non-mobile domains demonstrates agents' effectiveness in vulnerability detection [231, 247], malware analysis [118], and automated code annotation [102]. Mobile apps present additional complexity: extensive interactions

among components like UI widgets, callbacks, and communication channels—interactions that traditional work carefully models [73, 132]. Agents with memory could potentially maintain higher-level understanding of these global interactions, going beyond component-level analysis. While these foundational works provide promising directions, adapting these approaches to mobile-specific contexts is still in early stages.

Beyond these specific applications, agents' most distinctive capability lies in orchestrating existing mobile app analysis infrastructure. Rather than replacing individual analysis components, agents could autonomously plan and coordinate multiple tools—combining static analyzers, dynamic monitors, deobfuscation utilities, and semantic analysis—based on analysis objectives and intermediate findings. For instance, an agent analyzing privacy practices might: examine the manifest to identify relevant components, invoke taint analysis for specific flows, trigger dynamic testing to validate behaviors, cross-reference with policy text, and synthesize findings into structured reports. This autonomous tool orchestration represents a fundamentally different paradigm from embedding LLMs within individual analysis tools. Realizing this potential, however, requires addressing key challenges in agent architecture design, multi-tool coordination, and reliability guarantees. As agent technologies mature, we anticipate their integration with mobile app analysis infrastructure will enable more sophisticated and adaptive approaches, though ensuring reliability and controllability remains an open research question.

Overall Trends and Future Directions. These applications show consistent patterns in how LLMs are being integrated into mobile app analysis. LLMs primarily serve as components that bring semantic understanding to tasks traditionally requiring human expertise, such as interpreting user intent, evaluating behavior correctness, synthesizing insights from textual data, and, as agents, even orchestrating existing tools for intelligent analysis workflows. They are enabling analysis at substantial scales, from automating repetitive testing workflows to extracting patterns across entire app ecosystems.

Realizing the full potential of LLMs and agents in mobile app analysis requires addressing fundamental challenges. Behavioral nondeterminism and relatively high computational costs may limit their widespread adoption in production environments. How to enable more efficient, controllable, and cost-effective use of LLMs remains a critical research question. Future work should focus on developing practical frameworks that balance the power of LLMs with the reliability and cost-effectiveness required for industrial deployment. We anticipate that as these challenges are addressed, LLM-based approaches will become increasingly integral to mobile app analysis workflows, complementing rather than replacing traditional techniques.

6.1.2 LLM as Challenge.

While LLMs offer powerful capabilities for mobile app analysis, their integration into mobile applications simultaneously introduces new security and privacy risks that create fresh analytical challenges. These challenges manifest in two primary dimensions: the security and privacy implications of embedding LLM functionality within mobile apps, and the novel attack surfaces created when LLM-driven agents operate mobile applications autonomously.

LLM Integration Privacy and Security Challenges. Mobile apps increasingly embed LLM-backed features for content generation, recommendation, and conversational interaction [111]. This integration introduces vulnerabilities that traditional mobile app analysis techniques were not designed to detect. Research on non-mobile LLM applications has established the threat landscape: prompt injection attacks enable unauthorized access and content manipulation [37, 191], jailbreak techniques bypass safety mechanisms [67], environment manipulation exploits crafted content [140, 221], and privacy leakage occurs through training data extraction or contextual inference [65,

202]. Ecosystem-scale analysis shows widespread violations, with thousands of LLM-integrated applications collecting sensitive information against stated privacy policies [114].

Mobile environments amplify these risks due to the concentration of sensitive data, for instance, location history, biometric data, health records, financial credentials, and continuous sensor streams [81]. When mobile apps integrate LLM capabilities, user data may be processed through third-party API services or on-device models, creating new pathways for potential data leakage. Mobile-specific attack vectors compound these concerns: prompt injection through notification channels [76], manipulation via crafted GUI elements [218], and accessibility feature exploitation [81] can lead to unauthorized access or unintended data disclosure.

From an analysis perspective, these developments create fundamental challenges. Traditional mobile app analysis tools designed for deterministic logic face difficulties in tracking data flows through nondeterministic AI components, where the same input may produce varied outputs. Assessing privacy compliance becomes more complex when data processing involves opaque model inference, making it difficult to verify whether apps handle user data as claimed in privacy policies. Connecting observable AI behavior to specific code locations for vulnerability assessment requires new analytical approaches that bridge the gap between model-level operations and implementation-level code. While supply-chain defenses such as content filtering and pre-execution validation can mitigate certain attack vectors [74, 217], developing systematic analysis capabilities for verifying privacy compliance and detecting security vulnerabilities in AI-integrated mobile applications remains an important research direction as LLMs become more deeply embedded in mobile ecosystems.

Mobile Agent Attack Surfaces. LLM-driven agents that operate mobile applications autonomously [223] introduce attack surfaces where malicious apps exploit agent mechanisms. Non-mobile agent research has identified fundamental vulnerabilities: adversaries can manipulate agents through hidden prompts in content [37], malicious instructions via notifications [76], and jailbreak techniques including role-playing and adversarial suffixes [187, 250]. These attacks exploit the control-data confusion inherent in LLMs, where instructions and user content are processed through the same pathway.

In mobile environments, these vulnerabilities carry amplified consequences due to broader system access and pervasive device usage. Empirical evaluations show alarming success rates: touch-guided jailbreak attacks achieved over 80% success against state-of-the-art mobile agents [86]. Attackers exploit mobile-specific vectors including crafted UI elements with invisible text, poisoned deep links, and manipulated screenshots [86, 218] to induce agents to grant dangerous permissions, execute unauthorized operations, or perform cross-application attacks [218]. The systemic challenge arises from agents' dual vulnerability: their human-like operation patterns make them susceptible to traditional interface-based deception (phishing UIs, deceptive dialogs), while their LLM foundation exposes them to model-specific attacks like prompt injection and reasoning manipulation [218, 223].

This requires extending mobile app analysis beyond evaluating whether apps might deceive human users to assessing whether apps might manipulate autonomous agents. Systematic investigation of mobile agent attack methodologies, development of analysis techniques to identify agent-exploitable vulnerabilities in apps, and establishment of security standards for agent-enabled mobile applications remain critically underdeveloped as mobile operating systems increasingly integrate agent capabilities (e.g., Apple Intelligence [47], Google Gemini [106]).

6.2 Overall Implications of Mobile App Analysis Evolution

The challenges identified in §3–§5 affect mobile app analysis in different ways. To move from observing these challenges to understanding their implications, this section examines how analysis capabilities have changed, how requirements have evolved, and what stakeholders can do in response. This synthesis clarifies the practical significance of the transformations discussed earlier.

6.2.1 *Changes in Analysis Capabilities.*

Mobile app evolution creates substantial challenges for analysis, as discussed in previous sections. A critical issue is that code transparency faces increasing obstacles. Protection mechanisms, cross-platform frameworks, and multi-language compositions reduce what can be directly observed in packaged artifacts (§3). Runtime complexity further complicates analysis, AI components introduce nondeterministic behaviors and raise security and privacy concerns that lack systematic analytical approaches, while dynamic update mechanisms enable post-deployment behavior changes (§4).

At the same time, new analytical capabilities are emerging. LLM-based approaches bring semantic understanding to tasks that previously required manual effort or remained infeasible: analyzing obfuscated code, interpreting privacy policies, and connecting implementation behavior to policy descriptions for more direct verification (§6.1). Infrastructure improvements provide better foundations. Modern frameworks like Tai-e [194] offer more scalable, efficient, and accurate program analysis capabilities for mobile app analysis [73], while industrial tools such as Fastbot [61, 162] for automated GUI testing and AppShark [5] for static analysis continue to mature.

6.2.2 *Evolution of Analysis Requirements.*

The capability changes discussed above directly influence what analysis is expected to achieve. As traditional approaches face limitations, new scenarios emerge that demand different analytical strategies, while the scope of what needs to be analyzed expands beyond individual apps.

Regarding scenarios, requirements have diversified beyond traditional code verification. Agent-based interaction introduces risks where apps may manipulate autonomous agents rather than human users. Dynamic update mechanisms create needs for continuous monitoring rather than one-time pre-release assessment. Environmental awareness and context-dependent behaviors—such as detecting malicious actions triggered only in specific conditions—require runtime monitoring capabilities. Multi-modal interactions through voice, gesture, and AR/VR interfaces require simulation capabilities that traditional testing lacks. AI integration into mobile apps raises questions about behavioral nondeterminism and privacy implications, though systematic analytical approaches for these issues remain immature. These scenarios demand techniques tailored to their specific characteristics.

Regarding scope, traditional mobile app analysis focuses on individual applications in order to verify security properties, detecting vulnerabilities, and ensuring compliance within single apps. While this remains the core requirement, ecosystem-level analysis has become relevant: apps may behave differently across distribution channels, geographic markets, and regulatory jurisdictions. Apps providing similar functionality may exhibit different behaviors on diverse mobile device types, and analyzing these variations requires platform-specific infrastructure for systematic assessment. Effective analysis thus requires considering both individual app behavior and its ecosystem context.

6.2.3 *Implications for Stakeholders.*

The gaps between current capabilities and evolving requirements create challenges for different participants in the mobile ecosystem. Key stakeholders directly involved in mobile app analysis include developers who build and test apps, app markets and regulators who review and govern apps, and researchers who develop analysis techniques and infrastructure. Throughout previous sections, we have discussed various analysis challenges and emerging approaches from researchers' perspectives. Here, we examine how these changes affect developers and markets/regulators, and what actions they can take in response.

Developers and App Vendors. Developers primarily use mobile app analysis for quality assurance during development and testing phases. They face increasing complexity in ensuring app quality as modern development introduces diverse technical components, each requiring specialized testing approaches. AI functionality integration further complicates quality assurance: nondeterministic model outputs make exhaustive state coverage difficult, and traditional testing tools lack support for

these emerging components. Current toolchains provide only partial coverage, as new frameworks and AI integrations outpace the development of corresponding testing infrastructure.

However, LLM-assisted development and testing workflows offer promising opportunities to address some of these gaps. As discussed in §6.1, LLM-guided exploration can intelligently navigate complex state spaces, agent-based testing can simulate sophisticated user interactions, and automated test generation can reduce manual effort. Developers who engage with these emerging techniques can benefit from improved testing efficiency, though establishing robust quality assurance processes for AI-integrated apps remains an evolving practice.

App Markets and Regulators. App Markets often use mobile app analysis to conduct security and privacy compliance reviews of apps before they are distributed. They face growing difficulties in pre-deployment review. As discussed in §3, the transparency of packaged artifacts is strongly reduced. Runtime behavior changes (§4) add further complications, as apps may behave differently after passing initial review. The same app may also exhibit inconsistent practices across geographic markets and distribution channels.

Therefore, proactive measures should be taken to address these difficulties. This involves actively researching and adopting adaptive analysis techniques that can handle code protection. On the other hand, given the difficulty of directly analyzing the application's code implementation, it is also worthwhile to actively explore alternative methods. For instance, as indicated in §4, exploring continuous and systematic dynamic monitoring that tracks post-deployment behavior becomes increasingly important. Additionally, analyzing metadata beyond code itself, or combining such metadata with partially extractable features from protected code, has become more valuable with the emergence of LLMs, as discussed in §6.1. These complementary approaches deserve continued exploration.

Areas where analytical capabilities fall short also often present policy gaps. For example, while runtime updates are prohibited and related SDKs are blacklisted, developers can still tamper with their apps and update applications using flexible frameworks or even their own custom-designed techniques. Therefore, it is crucial to combine effective analytical techniques with clear rules to quantify such tampering, thereby protecting user rights and maintaining the order of the application ecosystem. Meanwhile, existing regulatory frameworks do not yet fully address emerging technologies like autonomous agents and embedded AI. Regulators may need to update existing frameworks to address emerging technologies.

7 Conclusion

By examining mobile app analysis through a three-layer perspective, this paper has provided a structured way to understand how modern apps challenge established methods. We have outlined how heterogeneous artifacts, increasingly complex runtime behaviors, and ecosystem-level fragmentation compound across layers, creating difficulties that traditional static and dynamic approaches struggle to address.

To respond, we have proposed a layered framework that not only structures the space of challenges but also highlights recurring themes: the loss of analyzability in packaged artifacts, the explosion of runtime complexity, and the difficulty of maintaining consistency across fragmented platforms and regulatory contexts. Building on this, we have identified research directions, from framework-aware analysis and adaptive testing to policy-aware compliance verification and ecosystem-scale comparison, that can guide future efforts.

By clarifying where current methods encounter limitations and pointing toward ways of combining targeted adaptation with unifying principles, our work aims to provide both researchers and practitioners with a reference point for navigating this evolving field. We hope that this perspective can support the development of more resilient and forward-looking approaches to mobile app analysis, and we look forward to future contributions that build upon and extend these directions.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (grants No.62572209, 62502168) and the Hubei Provincial Key Research and Development Program (grant No. 2025BAB057).

References

[1] 2025. 360 Jiagu. https://jiagu.360.cn/. Accessed: 2025-09-29.
[2] 2025. Android Automotive OS. https://source.android.com/docs/automotive/start/what_automotive. Accessed: 2025-02-21.
[3] 2025. Appdome — Mobile App Security & Protection Platform. https://www.appdome.com/. Accessed: 2025-09-29.
[4] 2025. Apple VisionOS. https://developer.apple.com/visionos/. Accessed: 2025-02-21.
[5] 2025. Appshark. https://github.com/bytedance/appshark. Accessed: 2025-09-29.
[6] 2025. Apptimize. https://apptimize.com/product/instant-update. Accessed: 2025-02-21.
[7] 2025. ArkTS Language - Getting Started. https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V13/arkts-get-started-V13 Accessed: 2025-02-21.
[8] 2025. Car app quality. https://developer.android.com/docs/quality-guidelines/car-app-quality. Accessed: 2025-02-21.
[9] 2025. CodePush. https://appcenter.ms/ Accessed: 2025-02-21.
[10] 2025. ColorOS 15. https://www.oppo.com/en/coloros15/. Accessed: 2025-09-29.
[11] 2025. Dart Programming Language. https://dart.dev/ Accessed: 2025-02-21.
[12] 2025. Flutter. https://flutter.dev/ Accessed: 2025-02-21.
[13] 2025. Flutter eval. https://pub.dev/packages/flutter_eval Accessed: 2025-02-21.
[14] 2025. Fuchsia. https://fuchsia.dev/ Accessed: 2025-02-21.
[15] 2025. GooglePlay Developer Policy Center. https://play.google/developer-content-policy/. Accessed: 2025-02-21.
[16] 2025. Hansel. https://hansel.io/. Accessed: 2025-02-21.
[17] 2025. HarmonyOS. https://www.harmonyos.com/en/ Accessed: 2025-02-21.
[18] 2025. HUAWEI AppGallery. https://consumer.huawei.com/en/mobileservices/appgallery/. Accessed: 2025-09-29.
[19] 2025. Ionic Framework. https://ionicframework.com/ Accessed: 2025-02-21.
[20] 2025. Kotlin Multiplatform Documentation. https://kotlinlang.org/docs/multiplatform.html Accessed: 2025-02-21.
[21] 2025. Kotlin Programming Language. https://kotlinlang.org/ Accessed: 2025-02-21.
[22] 2025. LangChain. https://www.langchain.com/ Accessed: 2025-02-21.
[23] 2025. Meta Quest. https://www.meta.com/quest/. Accessed: 2025-02-21.
[24] 2025. Mobile Security Armor, MSA. https://www.aliyun.com/product/mobilepaas/mpaas_ppm_public_cn. Accessed: 2025-09-29.
[25] 2025. Promon — App Shielding and Application Security. https://promon.io/. Accessed: 2025-09-29.
[26] 2025. React Native. https://rn.nodejs.cn/ Accessed: 2025-02-21.
[27] 2025. ShoreBird. https://shorebird.dev/ Accessed: 2025-02-21.
[28] 2025. Storyboard: Guides and sample code. https://developer.apple.com/library/content/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html. Accessed: 2025-02-21.
[29] 2025. Tencent Appstore. https://appstore.tencent.com/. Accessed: 2025-09-29.
[30] 2025. Tencent Mobile Security / LeGu). https://www.tencentcloud.com/zh/products/ms. Accessed: 2025-09-29.
[31] 2025. Tizen. https://www.tizen.org/ Accessed: 2025-02-21.
[32] 2025. TJ watson libraries for analysis. https://github.com/wala/WALA. Accessed: 2025-02-17.
[33] 2025. Ubuntu Touch. https://www.ubuntu-touch.io/apps/ Accessed: 2025-02-21.
[34] 2025. Wear OS app quality. https://developer.android.com/docs/quality-guidelines/wear-app-quality. Accessed: 2025-02-21.
[35] 2025. Xiaomi HyperOS. https://hyperos.mi.com/. Accessed: 2025-09-29.
[36] 2025. YASA Security Station. https://cybersec.antgroup.com/station.
[37] Sahar Abdelnabi, Kai Greshake, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, AISec 2023, Copenhagen, Denmark, 30 November 2023*, Maura Pintor, Xinyun Chen, and Florian Tramèr (Eds.). ACM, 79–90. doi:10.1145/3605764.3623985
[38] Maqsood Ahmad, Valerio Costamagna, Bruno Crispo, Francesco Bergadano, and Yury Zhauniarovich. 2020. StaDART: Addressing the problem of dynamic code updates in the security analysis of android applications. *J. Syst. Softw.* 159 (2020). doi:10.1016/J.JSS.2019.07.088
[39] Khaled Ahmed, Yingying Wang, Mieszko Lis, and Julia Rubin. 2023. ViaLin: Path-Aware Dynamic Taint Analysis for Android. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the*

Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023, Satish Chandra, Kelly Blincoe, and Paolo Tonella (Eds.). ACM, 1598–1610. doi:10.1145/3611643.3616330

[40] Omer Akgul, Sai Teja Peddinti, Nina Taft, Michelle L. Mazurek, Hamza Harkous, Animesh Srivastava, and Benoit Seguin. 2024. A Decade of Privacy-Relevant Android App Reviews: Large Scale Trends. *CoRR* abs/2403.02292 (2024). arXiv:2403.02292 doi:10.48550/ARXIV.2403.02292

[41] Faridah Akinotcho, Lili Wei, and Julia Rubin. 2025. Mobile Application Coverage: The 30% Curse and Ways Forward. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2751–2763. doi:10.1109/ICSE55347.2025.00142

[42] Marco Alecci, Jordan Samhi, Tegawendé F. Bissyandé, and Jacques Klein. 2025. Evaluating Large Language Models in detecting Secrets in Android Apps. *CoRR* abs/2510.18601 (2025). arXiv:2510.18601 doi:10.48550/ARXIV.2510.18601

[43] Alibaba. 2025. AndFix - Android App Hot - fix Library. https://github.com/alibaba/AndFix. Accessed: 2025-02-21.

[44] Mário Almeida, Muhammad Bilal, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Matteo Varvello, and Jeremy Blackburn. 2018. CHIMP: Crowdsourcing Human Inputs for Mobile Phones. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 45–54. doi:10.1145/3178876.3186035

[45] Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer. 2019. DL-Droid: Deep learning based android malware detection using real devices. *CoRR* abs/1911.10113 (2019). arXiv:1911.10113 http://arxiv.org/abs/1911.10113

[46] Benjamin Andow, Akhil Acharya, Dengfeng Li, William Enck, Kapil Singh, and Tao Xie. 2017. UiRef: analysis of sensitive user inputs in Android applications. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec 2017, Boston, MA, USA, July 18-20, 2017*, Guevara Noubir, Mauro Conti, and Sneha Kumar Kasera (Eds.). ACM, 23–34. doi:10.1145/3098243.3098247

[47] Apple Inc. 2024. Apple Intelligence is available today on iPhone, iPad, and Mac. https://www.apple.com/newsroom/2024/10/apple-intelligence-is-available-today-on-iphone-ipad-and-mac/. Accessed: 2025-01-10.

[48] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society. https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket

[49] Saba Arshad, Munam Ali Shah, Abdul Wahid, Amjad Mehmood, Houbing Song, and Hongnian Yu. 2018. SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System. *IEEE Access* 6 (2018), 4321–4339. doi:10.1109/ACCESS.2018.2792941

[50] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick D. McDaniel. 2014. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 259–269. doi:10.1145/2594291.2594299

[51] Alexandre Bartel, Jacques Klein, Martin Monperrus, and Yves Le Traon. 2014. Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges And Solutions for Analyzing Android. *CoRR* abs/1408.3976 (2014). arXiv:1408.3976 http://arxiv.org/abs/1408.3976

[52] Supraja Baskaran, Lianying Zhao, Mohammad Mannan, and Amr M. Youssef. 2023. Measuring the Leakage and Exploitability of Authentication Secrets in Super-apps: The WeChat Case. *CoRR* abs/2307.09317 (2023). arXiv:2307.09317 doi:10.48550/ARXIV.2307.09317

[53] Leonid Batyuk, Markus Herpich, Seyit Ahmet Çamtepe, Karsten Raddatz, Aubrey-Derrick Schmidt, and Sahin Albayrak. 2011. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. In *6th International Conference on Malicious and Unwanted Software, MALWARE 2011, Fajardo, Puerto Rico, USA, October 18-19, 2011*. IEEE Computer Society, 66–72. doi:10.1109/MALWARE.2011.6112328

[54] Manuel Benz, Erik Krogh Kristensen, Linghui Luo, Nataniel P. Borges Jr., Eric Bodden, and Andreas Zeller. 2021. Heaps'n Leaks: How Heap Snapshots Improve Android Taint Analysis. In *Software Engineering 2021, Fachtagung des GI-Fachbereichs Softwaretechnik, 22.-26. Februar 2021, Braunschweig/Virtuell (LNI, Vol. P-310)*, Anne Koziolek, Ina Schaefer, and Christoph Seidl (Eds.). Gesellschaft für Informatik e.V., 23–26. doi:10.18420/SE2021_02

[55] Stefano Berlato and Mariano Ceccato. 2020. A large-scale study on the adoption of anti-debugging and anti-tampering protections in android apps. *J. Inf. Secur. Appl.* 52 (2020), 102463. doi:10.1016/J.JISA.2020.102463

[56] Mario Luca Bernardi, Marta Cimitile, Damiano Distante, Fabio Martinelli, and Francesco Mercaldo. 2019. Dynamic malware detection and phylogeny analysis using process mining. *Int. J. Inf. Sec.* 18, 3 (2019), 257–284. doi:10.1007/S10207-018-0415-3

[57] David Beste, Grégoire Menguy, Hossein Hajipour, Mario Fritz, Antonio Emanuele Cinà, Sébastien Bardin, Thorsten Holz, Thorsten Eisenhofer, and Lea Schönherr. 2025. Exploring the Potential of LLMs for Code Deobfuscation. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 22nd International Conference, DIMVA 2025, Graz,*

Austria, July 9-11, 2025, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 15747), Manuel Egele, Veelasha Moonsamy, Daniel Gruss, and Michele Carminati (Eds.). Springer, 267–286. doi:10.1007/978-3-031-97620-9_15

[58] Benyamin Beyzaei, Saghar Talebipour, Ghazal Rafiei, Nenad Medvidovic, and Sam Malek. 2024. Automated Test Transfer Across Android Apps Using Large Language Models. *CoRR* abs/2411.17933 (2024). arXiv:2411.17933 doi:10.48550/ARXIV.2411.17933

[59] Parnika Bhat, Sunny Behal, and Kamlesh Dutta. 2023. A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. *Comput. Secur.* 130 (2023), 103277. doi:10.1016/J.COSE.2023.103277

[60] Benjamin Bichsel, Veselin Raychev, Petar Tsankov, and Martin T. Vechev. 2016. Statistical Deobfuscation of Android Applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 343–355. doi:10.1145/2976749.2978422

[61] ByteDance. 2025. Fastbot(2.0) is a model-based testing tool for modeling GUI transitions to discover app stability problems. https://github.com/bytedance/Fastbot_Android. Accessed: 2026-01-01.

[62] Haipeng Cai, Xiaoqin Fu, and Abdelwahab Hamou-Lhadj. 2020. A study of run-time behavioral evolution of benign versus malicious apps in android. *Inf. Softw. Technol.* 122 (2020), 106291. doi:10.1016/J.INFSOF.2020.106291

[63] Haipeng Cai, Na Meng, Barbara G. Ryder, and Danfeng Yao. 2019. DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling. *IEEE Trans. Inf. Forensics Secur.* 14, 6 (2019), 1455–1470. doi:10.1109/TIFS.2018.2879302

[64] Shaoheng Cao, Minxue Pan, Yuanhong Lan, and Xuandong Li. 2025. Intention-Based GUI Test Migration for Mobile Apps using Large Language Models. *Proc. ACM Softw. Eng.* 2, ISSTA (2025), 2296–2318. doi:10.1145/3728978

[65] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B. Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. 2020. Extracting Training Data from Large Language Models. *CoRR* abs/2012.07805 (2020). arXiv:2012.07805 https://arxiv.org/abs/2012.07805

[66] Patrick PF Chan, Lucas CK Hui, and Siu-Ming Yiu. 2012. Droidchecker: analyzing android applications for capability leak. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. 125–136.

[67] Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J. Pappas, Florian Tramèr, Hamed Hassani, and Eric Wong. 2024. JailbreakBench: An Open Robustness Benchmark for Jailbreaking Large Language Models. *CoRR* abs/2404.01318 (2024). arXiv:2404.01318 doi:10.48550/ARXIV.2404.01318

[68] Daihang Chen, Yonghui Liu, Mingyi Zhou, Yanjie Zhao, Haoyu Wang, Shuai Wang, Xiao Chen, Tegawendé F. Bissyandé, Jacques Klein, and Li Li. 2024. LLM for Mobile: An Initial Roadmap. *CoRR* abs/2407.06573 (2024). arXiv:2407.06573 doi:10.48550/ARXIV.2407.06573

[69] Guoqiang Chen, Xin Jin, and Zhiqiang Lin. 2025. JsDeObsBench: Measuring and Benchmarking LLMs for JavaScript Deobfuscation. *CoRR* abs/2506.20170 (2025). arXiv:2506.20170 doi:10.48550/ARXIV.2506.20170

[70] Haonan Chen, Daihang Chen, Yonghui Liu, Xiaoyu Sun, and Li Li. 2024. Are Your Android App Analyzers Still Relevant?. In *Proceedings of the IEEE/ACM 11th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2024, Lisbon, Portugal, April 14-15, 2024*, Denys Poshyvanyk, Gemma Catolino, Julia Rubin, and Wing Lam (Eds.). ACM, 69–73. doi:10.1145/3647632.3651388

[71] Haonan Chen, Daihang Chen, Yizhuo Yang, Lingyun Xu, Liang Gao, Mingyi Zhou, Chunming Hu, and Li Li. 2025. ArkAnalyzer: The Static Analysis Framework for OpenHarmony. *CoRR* abs/2501.05798 (2025). arXiv:2501.05798 doi:10.48550/ARXIV.2501.05798

[72] Kai Chen, Peng Liu, and Yingjun Zhang. 2014. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 175–186. doi:10.1145/2568225.2568286

[73] Menglong Chen, Tian Tan, Minxue Pan, and Yue Li. 2025. PacDroid: A Pointer-Analysis-Centric Framework for Security Vulnerabilities in Android Apps. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2803–2815. doi:10.1109/ICSE55347.2025.00208

[74] Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David A. Wagner, and Chuan Guo. 2025. SecAlign: Defending Against Prompt Injection with Preference Optimization. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM, 2833–2847. doi:10.1145/3719027.3744836

[75] Wenhao Chen, Morris Chang, Witawas Srisa-an, and Yong Guan. 2025. APEX: Automatic Event Sequence Generation for Android Applications. *CoRR* abs/2509.02412 (2025). arXiv:2509.02412 doi:10.48550/ARXIV.2509.02412

[76] Yurun Chen, Xueyu Hu, Keting Yin, Juncheng Li, and Shengyu Zhang. 2025. AEIA-MN: Evaluating the Robustness of Multimodal LLM-Powered Mobile Agents Against Active Environmental Injection Attacks. *CoRR* abs/2502.13053 (2025). arXiv:2502.13053 doi:10.48550/ARXIV.2502.13053

[77] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 429–440. doi:10.1109/ASE.2015.89

[78] Chenhui Cui, Tao Li, Junjie Wang, Chunyang Chen, Dave Towey, and Rubing Huang. 2024. Large Language Models for Mobile GUI Text Input Generation: An Empirical Study. *CoRR* abs/2404.08948 (2024). arXiv:2404.08948 doi:10.48550/ARXIV.2404.08948

[79] Hao Cui, Rahmadi Trimananda, Athina Markopoulou, and Scott Jordan. 2022. PoliGraph: Automated Privacy Policy Analysis using Knowledge Graphs. *CoRR* abs/2210.06746 (2022). arXiv:2210.06746 doi:10.48550/ARXIV.2210.06746

[80] Xingmin Cui, Jingxuan Wang, Lucas Chi Kwong Hui, Zhongwei Xie, Tian Zeng, and Siu-Ming Yiu. 2015. WeChecker: efficient and precise detection of privilege escalation vulnerabilities in Android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks, New York, NY, USA, June 22-26, 2015*. ACM, 25:1–25:12. doi:10.1145/2766498.2766509

[81] Jorge Damian. 2025. The OWASP AI/LLM Top 10: Understanding Security and Privacy Risks in AI-Powered Mobile Applications. https://www.nowsecure.com/blog/2025/11/05/the-owasp-ai-llm-top-10-understanding-security-and-privacy-risks-in-ai-powered-mobile-applications/. NowSecure blog, Accessed: 2025-12-22.

[82] Gianni D'Angelo, Massimo Ficco, and Francesco Palmieri. 2020. Malware detection in mobile environments based on Autoencoders and API-images. *J. Parallel Distributed Comput.* 137 (2020), 26–33. doi:10.1016/J.JPDC.2019.11.001

[83] Jiapeng Deng, Tianming Liu, Yanjie Zhao, Chao Wang, Lin Zhang, and Haoyu Wang. 2025. Walls Have Ears: Demystifying Notification Listener Usage in Android Apps. *Proc. ACM Softw. Eng.* 2, ISSTA (2025), 434–456. doi:10.1145/3728898

[84] Anthony Desnos. 2012. Android: Static Analysis Using Similarity Distance. In *45th Hawaii International International Conference on Systems Science (HICSS-45 2012), Proceedings, 4-7 January 2012, Grand Wailea, Maui, HI, USA*. IEEE Computer Society, 5394–5403. doi:10.1109/HICSS.2012.114

[85] Michalis Diamantaris, Elias P. Papadopoulos, Evangelos P. Markatos, Sotiris Ioannidis, and Jason Polakis. 2019. REAPER: Real-time App Analysis for Augmenting the Android Permission System. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019*, Gail-Joon Ahn, Bhavani Thuraisingham, Murat Kantarcioglu, and Ram Krishnan (Eds.). ACM, 37–48. doi:10.1145/3292006.3300027

[86] Renhua Ding, Xiao Yang, Zhengwei Fang, Jun Luo, Kun He, and Jun Zhu. 2025. Practical and Stealthy Touch-Guided Jailbreak Attacks on Deployed Mobile Vision-Language Agents. arXiv:2510.07809 [cs.CR] https://arxiv.org/abs/2510.07809

[87] Shuaike Dong, Menghao Li, Wenrui Diao, Xiangyu Liu, Jian Liu, Zhou Li, Fenghao Xu, Kai Chen, Xiaofeng Wang, and Kehuan Zhang. 2018. Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild. *CoRR* abs/1801.01633 (2018). arXiv:1801.01633 http://arxiv.org/abs/1801.01633

[88] Zhen Dong, Marcel Böhme, Lucia Cojocaru, and Abhik Roychoudhury. 2020. Time-travel testing of Android apps. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 481–492. doi:10.1145/3377811.3380402

[89] Zikan Dong, Yanjie Zhao, Tianming Liu, Chao Wang, Guosheng Xu, Guoai Xu, Lin Zhang, and Haoyu Wang. 2024. Same App, Different Behaviors: Uncovering Device-specific Behaviors in Android Apps. In *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2099–2109.

[90] Yue Duan, Mu Zhang, Abhishek Vasisht Bhaskar, Heng Yin, Xiaorui Pan, Tongxin Li, Xueqiang Wang, and XiaoFeng Wang. 2018. Things You May Not Know About Android (Un)Packers: A Systematic Study based on Whole-System Emulation. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society. https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_04A-4_Duan_paper.pdf

[91] Shouki Ebad and Abdulbasit Darem. 2024. Exploring Android Obfuscators and Deobfuscators: An Empirical Investigation. *Electronics* 13 (06 2024), 2272. doi:10.3390/electronics13122272

[92] Filipe Espósito. 2024. A deep dive into how developers trick App Store review into approving malicious apps. *9to5Mac* (2024). https://9to5mac.com/2024/08/02/developers-trick-app-store-review/ Detailed analysis of pirate streaming apps using geofencing and Microsoft CodePush SDK to bypass Apple's App Store review process.

[93] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. *CoRR* abs/2310.03533 (2023). arXiv:2310.03533 doi:10.48550/ARXIV.2310.03533

[94] Ming Fan, Xiapu Luo, Jun Liu, Meng Wang, Chunyin Nong, Qinghua Zheng, and Ting Liu. 2019. Graph embedding based familial analysis of Android malware using unsupervised learning. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 771–782. doi:10.1109/ICSE.2019.00085

[95] Pengbin Feng, Jianfeng Ma, Cong Sun, Xinpeng Xu, and Yuwan Ma. 2018. A Novel Dynamic Android Malware Detection System With Ensemble Learning. *IEEE Access* 6 (2018), 30996–31011. doi:10.1109/ACCESS.2018.2844349

[96] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 67:1–67:13. doi:10.1145/3597503.3608137

[97] Sidong Feng, Changhao Du, Huaxiao Liu, Qingnan Wang, Zhengwei Lv, Mengfei Wang, and Chunyang Chen. 2025. Breaking Single-Tester Limits: Multi-Agent LLMs for Multi-User Feature Testing. *CoRR* abs/2506.17539 (2025). arXiv:2506.17539 doi:10.48550/ARXIV.2506.17539

[98] Sidong Feng, Haochuan Lu, Jianqin Jiang, Ting Xiong, Likun Huang, Yinglin Liang, Xiaoqin Li, Yuetang Deng, and Aldeida Aleti. 2024. Enabling Cost-Effective UI Automation Testing with Retrieval-Based LLMs: A Case Study in WeChat. *CoRR* abs/2409.07829 (2024). arXiv:2409.07829 doi:10.48550/ARXIV.2409.07829

[99] Jyoti Gajrani, Umang Agarwal, Vijay Laxmi, Bruhadeshwar Bezawada, Manoj Singh Gaur, Meenakshi Tripathi, and Akka Zemmari. 2020. EspyDroid+: Precise reflection analysis of android apps. *Comput. Secur.* 90 (2020), 101688. doi:10.1016/J.COSE.2019.101688

[100] GDPR.eu. 2024. General Data Protection Regulation (GDPR) Compliance Guidelines. MISC. [Online]. Available: https://gdpr.eu/.

[101] Xiuting Ge, Ya Pan, Yong Fan, and Chunrong Fang. 2019. AMDroid: Android Malware Detection Using Function Call Graphs. In *19th IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2019, Sofia, Bulgaria, July 22-26, 2019*. IEEE, 71–77. doi:10.1109/QRS-C.2019.00027

[102] Ashutosh Ghimire, Sahasra Rao Lingala, Junjie Zhang, Faris Alsulami, and Fathi Amsaad. 2025. A Survey on Application of AI on Reverse Engineering for Software Analysis and Security. *IEEE Access* 13 (2025), 152903–152913. doi:10.1109/ACCESS.2025.3593456

[103] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. 2012. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *Trust and Trustworthy Computing - 5th International Conference, TRUST 2012, Vienna, Austria, June 13-15, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7344)*, Stefan Katzenbeisser, Edgar R. Weippl, L. Jean Camp, Melanie Volkamer, Mike K. Reiter, and Xinwen Zhang (Eds.). Springer, 291–307. doi:10.1007/978-3-642-30921-2_17

[104] Igor Golovko, Oleg Savenko, Petro Vizhevskiy, Olexandr Klein, and Abdel-Badeeh M. Salem. 2024. Obfuscation technologies of high-level source code using artificial intelligence. In *Proceedings of the 1st International Workshop on Intelligent & CyberPhysical Systems (ICyberPhyS 2024), Khmelnytskyi, Ukraine, June 28, 2024 (CEUR Workshop Proceedings)*, Tetiana Hovorushchenko, Oleg Savenko, Peter T. Popov, and Sergii Lysenko (Eds.). CEUR-WS.org, 324–338. https://ceur-ws.org/Vol-3736/paper24.pdf

[105] Google. 2024. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey. Accessed: Feb. 14, 2024.

[106] Google. 2025. The Assistant experience on mobile is upgrading to Gemini. https://blog.google/products/gemini/google-assistant-gemini-mobile/. Accessed: 2025-01-10.

[107] Chenkai Guo, Qianlu Wang, Naipeng Dong, Lingling Fan, Tianhong Wang, Weijie Zhang, Enbao Chen, Zheli Liu, and Lu Yu. 2025. EP-Detector: Automatic Detection of Error-Prone Operation Anomalies in Android Applications. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2739–2750. doi:10.1109/ICSE55347.2025.00029

[108] Hanyang Guo, Hong-Ning Dai, Xiapu Luo, Zibin Zheng, Gengyang Xu, and Fengliang He. 2024. An Empirical Study on Oculus Virtual Reality Applications: Security and Privacy Perspectives. *CoRR* abs/2402.13815 (2024). arXiv:2402.13815 doi:10.48550/ARXIV.2402.13815

[109] Jiawei Guo, Yu Nong, Zhiqiang Lin, and Haipeng Cai. 2025. Code Speaks Louder: Exploring Security and Privacy Relevant Regional Variations in Mobile Applications. In *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, Marina Blanton, William Enck, and Cristina Nita-Rotaru (Eds.). IEEE, 4284–4302. doi:10.1109/SP61157.2025.00225

[110] Yanhui Guo, Dong Wang, Liu Wang, Yongsheng Fang, Chao Wang, Minghui Yang, Tianming Liu, and Haoyu Wang. 2024. Beyond App Markets: Demystifying Underground Mobile App Distribution Via Telegram. *CoRR* abs/2408.03482 (2024). arXiv:2408.03482 doi:10.48550/ARXIV.2408.03482

[111] Kimberly Hau, Safwat Hassan, and Shurui Zhou. 2025. LLMs in Mobile Apps: Practices, Challenges, and Opportunities. *CoRR* abs/2502.15908 (2025). arXiv:2502.15908 doi:10.48550/ARXIV.2502.15908

[112] Android Headlines. 2024. This is how malicious apps trick Apple's App Store security filters. *Android Headlines* (2024). https://www.androidheadlines.com/2024/08/malicious-apps-apple-app-store-security-filters.html

[113] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John C. Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. *CoRR* abs/2308.10620 (2023). arXiv:2308.10620 doi:10.48550/ARXIV.2308.10620

[114] Xinyi Hou, Yanjie Zhao, and Haoyu Wang. 2024. On the (In)Security of LLM App Stores. *CoRR* abs/2407.08422 (2024). arXiv:2407.08422 doi:10.48550/ARXIV.2407.08422

[115] Peiwei Hu, Ruigang Liang, and Kai Chen. 2024. DeGPT: Optimizing Decompiler Output with LLM. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/degpt-optimizing-decompiler-output-with-llm/

[116] Yuchao Huang, Junjie Wang, Zhe Liu, Yawen Wang, Song Wang, Chunyang Chen, Yuanzhe Hu, and Qing Wang. 2023. CrashTranslator: Automatically Reproducing Mobile Application Crashes Directly from Stack Trace. *CoRR* abs/2310.07128 (2023). arXiv:2310.07128 doi:10.48550/ARXIV.2310.07128

[117] Syed Fatiul Huq, Mahan Tafreshipour, Kate Kalcevich, and Sam Malek. 2025. Automated Generation of Accessibility Test Reports from Recorded User Transcripts. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 204–216. doi:10.1109/ICSE55347.2025.00043

[118] Hamed Jelodar, Samita Bai, Parisa Hamedi, Hesamodin Mohammadian, Roozbeh Razavi-Far, and Ali A. Ghorbani. 2025. Large Language Model (LLM) for Software Security: Code Analysis, Malware Analysis, Reverse Engineering. *CoRR* abs/2504.07137 (2025). arXiv:2504.07137 doi:10.48550/ARXIV.2504.07137

[119] Ran Jin, Liu Wang, Shidong Pan, Luona Xu, Tianming Liu, and Haoyu Wang. 2026. Understanding User Privacy Perceptions of GenAI Smartphones. arXiv:2604.05571 [cs.CR] https://arxiv.org/abs/2604.05571

[120] Bangyan Ju, Jin Yang, Tingting Yu, Tamerlan Abdullayev, Yuanyuan Wu, Dingbang Wang, and Yu Zhao. 2024. A Study of Using Multimodal LLMs for Non-Crash Functional Bug Detection in Android Apps. *CoRR* abs/2407.19053 (2024). arXiv:2407.19053 doi:10.48550/ARXIV.2407.19053

[121] Shujahat Ali Khan, Hasan Raza Bajwa, Jawahar Sundaram, Pritika, and Bharanidharan Shanmugam. 2024. Vulnerability Analysis and Exploitation Attacks on Smart Wearable Devices. In *2024 2nd International Conference on Advancement in Computation & Computer Technologies (InCACCT)*. 911–916. doi:10.1109/InCACCT61598.2024.10550999

[122] Mst. Eshita Khatun, Lamine Noureddine, Zhiyong Sui, and Aisha I. Ali-Gombe. 2025. AndroByte: LLM-Driven Privacy Analysis through Bytecode Summarization and Dynamic Dataflow Call Graph Generation. *CoRR* abs/2510.15112 (2025). arXiv:2510.15112 doi:10.48550/ARXIV.2510.15112

[123] Hanna Kim, Minkyoo Song, Seung Ho Na, Seungwon Shin, and Kimin Lee. 2024. When LLMs Go Online: The Emerging Threat of Web-Enabled LLMs. *CoRR* abs/2410.14569 (2024). arXiv:2410.14569 doi:10.48550/ARXIV.2410.14569

[124] Patrick Kochberger, Sebastian Schrittwieser, Bart Coppens, and Bjorn De Sutter. 2023. Evaluation Methodologies in Software Protection Research. *CoRR* abs/2307.07300 (2023). arXiv:2307.07300 doi:10.48550/ARXIV.2307.07300

[125] Patrick Kochberger, Sebastian Schrittwieser, Stefan Schweighofer, Peter Kieseberg, and Edgar R. Weippl. 2021. SoK: Automatic Deobfuscation of Virtualization-protected Applications. In *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, Delphine Reinhardt and Tilo Müller (Eds.). ACM, 6:1–6:15. doi:10.1145/3465481.3465772

[126] Renuka Kumar, Apurva Virkud, Ram Sundara Raman, Atul Prakash, and Roya Ensafi. 2022. A Large-scale Investigation into Geodifferences in Mobile Apps. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 1203–1220. https://www.usenix.org/conference/usenixsecurity22/presentation/kumar

[127] Patrick Lam, Eric Bodden, Ondrej Lhoták, and Laurie Hendren. 2011. The Soot framework for Java program analysis: a retrospective. In *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, Vol. 15.

[128] Yeonjoon Lee, Xueqiang Wang, Kwangwuk Lee, Xiaojing Liao, XiaoFeng Wang, Tongxin Li, and Xianghang Mi. 2019. Understanding iOS-based Crowdturfing Through Hidden UI Analysis. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 765–781. https://www.usenix.org/conference/usenixsecurity19/presentation/lee

[129] Haodong Li, Xiao Cheng, Guohan Zhang, Guosheng Xu, Guoai Xu, and Haoyu Wang. 2025. Mitigating Emergent Malware Label Noise in DNN-Based Android Malware Detection. *Proc. ACM Softw. Eng.* 2, FSE (2025), 1136–1159. doi:10.1145/3715769

[130] Haodong Li, Xiao Cheng, Yanjie Zhao, Guosheng Xu, Guoai Xu, and Haoyu Wang. 2025. Understanding Model Weaknesses: A Path to Strengthening DNN-Based Android Malware Detection. *Proc. ACM Softw. Eng.* 2, ISSTA (2025), 320–342. doi:10.1145/3728884

[131] Heng Li, Zhang Cheng, Bang Wu, Liheng Yuan, Cuiying Gao, Wei Yuan, and Xiapu Luo. 2023. Black-box Adversarial Example Attack towards FCG Based Android Malware Detection under Incomplete Feature Information. *CoRR* abs/2303.08509 (2023). arXiv:2303.08509 doi:10.48550/ARXIV.2303.08509

[132] Li Li, Alexandre Bartel, Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick D. McDaniel. 2015. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, 280–291. doi:10.1109/ICSE.2015.48

[133] Li Li, Alexandre Bartel, Jacques Klein, and Yves Le Traon. 2014. Automatically Exploiting Potential Component Leaks in Android Applications. In *13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014, Beijing, China, September 24-26, 2014*. IEEE Computer Society, 388–397. doi:10.1109/TRUSTCOM.2014.50

[134] Shuai Li, Zhemin Yang, Guangliang Yang, Hange Zhang, Nan Hua, Yurui Huang, and Min Yang. 2023. Notice the Imposter! A Study on User Tag Spoofing Attack in Mobile Apps. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 5485–5501. https://www.usenix.org/conference/usenixsecurity23/presentation/li-shuai

[135] Wei Li, Borui Yang, Hangyu Ye, Liyao Xiang, Qingxiao Tao, Xinbing Wang, and Chenghu Zhou. 2024. MiniTracker: Large-Scale Sensitive Information Tracking in Mini Apps. *IEEE Trans. Dependable Secur. Comput.* 21, 4 (2024), 2099–2114. doi:10.1109/TDSC.2023.3299945

[136] Yao Li, Sen Fang, Tao Zhang, and Haipeng Cai. 2024. Enhancing Android Malware Detection: The Influence of ChatGPT on Decision-centric Task. *CoRR* abs/2410.04352 (2024). arXiv:2410.04352 doi:10.48550/ARXIV.2410.04352

[137] Youwei Li, Yangyang Li, and Yangzhao Yang. 2024. Test-Agent: A Multimodal App Automation Testing Framework Based on the Large Language Model. In *4th IEEE International Conference on Digital Twins and Parallel Intelligence, DTPI 2024, Wuhan, China, October 18-20, 2024*. IEEE, 609–614. doi:10.1109/DTPI61353.2024.10778901

[138] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: a lightweight UI-guided test input generator for Android. In *Proceedings of the 39th International Conference on Software Engineering Companion* (Buenos Aires, Argentina) *(ICSE-C '17)*. IEEE Press, 23–26. doi:10.1109/ICSE-C.2017.8

[139] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2019. Humanoid: A Deep Learning-Based Approach to Automated Black-box Android App Testing. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 1070–1073. doi:10.1109/ASE.2019.00104

[140] Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. 2024. EIA: Environmental Injection Attack on Generalist Web Agents for Privacy Leakage. *CoRR* abs/2409.11295 (2024). arXiv:2409.11295 doi:10.48550/ARXIV.2409.11295

[141] Martina Lindorfer, Stamatis Volanis, Alessandro Sisto, Matthias Neugschwandtner, Elias Athanasopoulos, Federico Maggi, Christian Platzer, Stefano Zanero, and Sotiris Ioannidis. 2014. AndRadar: Fast Discovery of Android Applications in Alternative Markets. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8550)*, Sven Dietrich (Ed.). Springer, 51–71. doi:10.1007/978-3-319-08509-8_4

[142] Fengyu Liu, Yuan Zhang, Jiaqi Luo, Jiarun Dai, Tian Chen, Letian Yuan, Zhengmin Yu, Youkun Shi, Ke Li, Chengyuan Zhou, Hao Chen, and Min Yang. 2025. Make Agent Defeat Agent: Automatic Detection of Taint-Style Vulnerabilities in LLM-based Agents. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, Lujo Bauer and Giancarlo Pellegrino (Eds.). USENIX Association, 3767–3786. https://www.usenix.org/conference/usenixsecurity25/presentation/liu-fengyu

[143] Guangyi Liu, Pengxiang Zhao, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, Xiaoyu Liang, Wenhao Wang, Tianze Wu, Linghao Li, Hao Wang, Guanjing Xiong, Yong Liu, and Hongsheng Li. 2025. LLM-Powered GUI Agents in Phone Automation: Surveying Progress and Prospects. *CoRR* abs/2504.19838 (2025). arXiv:2504.19838 doi:10.48550/ARXIV.2504.19838

[144] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large Language Model-Based Agents for Software Engineering: A Survey. *CoRR* abs/2409.02977 (2024). arXiv:2409.02977 doi:10.48550/ARXIV.2409.02977

[145] Pei Liu, Terry Yue Zhuo, Jiawei Deng, Thong James, Shidong Pan, Sherry Xu, Zhenchang Xing, Qinghua Lu, Xiaoning Du, and Hongyu Zhang. 2025. LLMAID: Identifying AI Capabilities in Android Apps with LLMs. *CoRR* abs/2511.19059 (2025). arXiv:2511.19059 doi:10.48550/ARXIV.2511.19059

[146] Ruofan Liu, Xiwen Teoh, Yun Lin, Guanjie Chen, Ruofei Ren, Denys Poshyvanyk, and Jin Song Dong. 2025. GUIPilot: A Consistency-based Mobile GUI Testing Approach for Detecting Application-specific Bugs. *CoRR* abs/2506.07385 (2025). arXiv:2506.07385 doi:10.48550/ARXIV.2506.07385

[147] Xing Liu, Jiqiang Liu, Sencun Zhu, Wei Wang, and Xiangliang Zhang. 2020. Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem. *IEEE Trans. Mob. Comput.* 19, 5 (2020), 1184–1199. doi:10.1109/TMC.2019.2903186

[148] Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, Xinyi Liu, Xinying Chen, Xinyue Yang, Yang Yang, Yifan Xu, Yu Yang, Yujia Wang, Yulin Xu, Zehan Qi, Yuxiao Dong, and Jie Tang. 2024. AutoGLM: Autonomous Foundation Agents for GUIs. *CoRR* abs/2411.00820 (2024). arXiv:2411.00820 doi:10.48550/ARXIV.2411.00820

[149] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. 2015. When Good Becomes Evil: Keystroke Inference with Smartwatch. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 1273–1285. doi:10.1145/2810103.2813668

[150] Yonghui Liu, Xiao Chen, Pei Liu, John Grundy, Chunyang Chen, and Li Li. 2023. ReuNify: A Step Towards Whole Program Analysis for React Native Android Apps. *CoRR* abs/2309.03524 (2023). arXiv:2309.03524 doi:10.48550/ARXIV.2309.03524

[151] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2023. Prompt Injection attack against LLM-integrated Applications. *CoRR* abs/2306.05499 (2023). arXiv:2306.05499 doi:10.48550/ARXIV.2306.05499

[152] Yonghui Liu, Li Li, Pingfan Kong, Xiaoyu Sun, and Tegawendé F. Bissyandé. 2021. A First Look at Security Risks of Android TV Apps. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021 - Workshops, Melbourne, Australia, November 15-19, 2021*. IEEE, 59–64. doi:10.1109/ASEW52652.2021.00023

[153] Yang Liu, Chaoshun Zuo, Zonghua Zhang, Shanqing Guo, and Xin-Shun Xu. 2018. An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid Web apps. *World Wide Web* 21, 1 (2018), 127–150. doi:10.1007/S11280-017-0458-9

[154] Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. 2022. Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing. *CoRR* abs/2212.04732 (2022). arXiv:2212.04732 doi:10.48550/ARXIV.2212.04732

[155] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2023. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. *CoRR* abs/2310.15780 (2023). arXiv:2310.15780 doi:10.48550/ARXIV.2310.15780

[156] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2023. Testing the Limits: Unusual Text Inputs Generation for Mobile App Crash Detection with Large Language Model. *CoRR* abs/2310.15657 (2023). arXiv:2310.15657 doi:10.48550/ARXIV.2310.15657

[157] Zhe Liu, Cheng Li, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Yawen Wang, Jun Hu, and Qing Wang. 2025. Seeing is Believing: Vision-Driven Non-Crash Functional Bug Detection for Mobile Apps. *IEEE Trans. Software Eng.* 51, 12 (2025), 3452–3466. doi:10.1109/TSE.2025.3614469

[158] Zichen Liu and Xusheng Xiao. 2025. AppBDS: LLM-Powered Description Synthesis for Sensitive Behaviors in Mobile Apps. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, Seoul, South Korea.

[159] Andrea De Lorenzo, Fabio Martinelli, Eric Medvet, Francesco Mercaldo, and Antonella Santone. 2020. Visualizing the outcome of dynamic analysis of Android malware with VizMal. *J. Inf. Secur. Appl.* 50 (2020). doi:10.1016/J.JISA.2019.102423

[160] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. 2012. CHEX: statically vetting Android apps for component hijacking vulnerabilities. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM, 229–240. doi:10.1145/2382196.2382223

[161] Zhang Luoshi, Niu Yan, Wu Xiao, Wang Zhaoguo, and Xue Yibo. 2013. A3: automatic analysis of android malware. In *1st International Workshop on Cloud Computing and Information Security*. Atlantis Press, 89–93.

[162] Zhengwei Lv, Chao Peng, Zhao Zhang, Ting Su, Kai Liu, and Ping Yang. 2022. Fastbot2: Reusable Automated Model-based GUI Testing for Android Enhanced by Reinforcement Learning. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 135:1–135:5. doi:10.1145/3551349.3559505

[163] Zhuo Ma, Haoran Ge, Yang Liu, Meng Zhao, and Jianfeng Ma. 2019. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. *IEEE Access* 7 (2019), 21235–21245. doi:10.1109/ACCESS.2019.2896003

[164] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. LibRadar: fast and accurate detection of third-party libraries in Android apps. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 653–656. doi:10.1145/2889160.2889178

[165] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: multi-objective automated testing for Android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, Andreas Zeller and Abhik Roychoudhury (Eds.). ACM, 94–105. doi:10.1145/2931037.2931054

[166] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J. Ross, and Gianluca Stringhini. 2016. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. *CoRR* abs/1612.04433 (2016). arXiv:1612.04433 http://arxiv.org/abs/1612.04433

[167] Forough Mehralian, Ziyao He, and Sam Malek. 2025. Automated Accessibility Analysis of Dynamic Content Changes on Mobile Apps. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2689–2701. doi:10.1109/ICSE55347.2025.00039

[168] Microsoft. 2024. 6 AI trends you'll see more of in 2025. https://news.microsoft.com/source/features/ai/6-ai-trends-youll-see-more-of-in-2025/.

[169] Seyedreza Mohseni, Seyedali Mohammadi, Deepa Tilwani, Yash Saxena, Gerald Ndawula, Sriram Vema, Edward Raff, and Manas Gaur. 2024. Can LLMs Obfuscate Code? A Systematic Analysis of Large Language Models into Assembly Code Obfuscation. *CoRR* abs/2412.16135 (2024). arXiv:2412.16135 doi:10.48550/ARXIV.2412.16135

[170] Charlotte Moremen, Jordan Hoogsteden, and Eleanor Birrell. 2024. Generational Differences in Understandings of Privacy Terminology. *Proc. Priv. Enhancing Technol.* 2024, 3 (2024), 589–605. doi:10.56553/POPETS-2024-0094

[171] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. 2021. Share First, Ask Later (or Never?) Studying Violations of GDPR's Explicit Consent in Android Apps. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael D. Bailey and Rachel Greenstadt (Eds.). USENIX Association, 3667–3684. https://www.usenix.org/conference/usenixsecurity21/presentation/nguyen

[172] Akila Niroshan, Suranga Seneviratne, and Aruna Seneviratne. 2025. An Empirical Study of Code Obfuscation Practices in the Google Play Store. *CoRR* abs/2502.04636 (2025). arXiv:2502.04636 doi:10.48550/ARXIV.2502.04636

[173] Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement learning based curiosity-driven testing of Android applications. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, Sarfraz Khurshid and Corina S. Pasareanu (Eds.). ACM, 153–164. doi:10.1145/3395363.3397354

[174] Ya Pan, Xiuting Ge, Chunrong Fang, and Yong Fan. 2020. A Systematic Literature Review of Android Malware Detection Using Static Analysis. *IEEE Access* 8 (2020), 116363–116379. doi:10.1109/ACCESS.2020.3002842

[175] Achilleas Papageorgiou, Michael Strigkos, Eugenia A. Politou, Efthimios Alepis, Agusti Solanas, and Constantinos Patsakis. 2018. Security and Privacy Analysis of Mobile Health Applications: The Alarming State of Practice. *IEEE Access* 6 (2018), 9390–9403. doi:10.1109/ACCESS.2018.2799522

[176] Constantinos Patsakis, Fran Casino, and Nikolaos Lykousas. 2024. Assessing LLMs in Malicious Code Deobfuscation of Real-world Malware Campaigns. *CoRR* abs/2404.19715 (2024). arXiv:2404.19715 doi:10.48550/ARXIV.2404.19715

[177] Aleksandr Pilgun, Olga Gadyatskaya, Stanislav Dashevskyi, Yury Zhauniarovich, and Artsiom Kushniarou. 2018. Fine-grained Code Coverage Measurement in Automated Black-box Android Testing. *CoRR* abs/1812.10729 (2018). arXiv:1812.10729 http://arxiv.org/abs/1812.10729

[178] Siegfried Rasthofer, Steven Arzt, Enrico Lovat, and Eric Bodden. 2014. DroidForce: Enforcing Complex, Data-centric, System-wide Policies in Android. In *Ninth International Conference on Availability, Reliability and Security, ARES 2014, Fribourg, Switzerland, September 8-12, 2014*. IEEE Computer Society, 40–49. doi:10.1109/ARES.2014.13

[179] Rajkumar Singh Rathore, Chaminda Hewage, Omprakash Kaiwartya, and Jaime Lloret. 2022. In-Vehicle Communication Cyber Security: Challenges and Solutions. *Sensors* 22, 17 (2022), 6679. doi:10.3390/S22176679

[180] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 603–620. https://www.usenix.org/conference/usenixsecurity19/presentation/reardon

[181] Counterpoint Research. 2024. Global Smartphone Sales Share by Operating System. https://www.counterpointresearch.com/insights/global-smartphone-os-market-share/ Accessed: 2026-05-15.

[182] Andrea Romdhana, Alessio Merlo, Mariano Ceccato, and Paolo Tonella. 2021. Deep Reinforcement Learning for Black-Box Testing of Android Apps. *CoRR* abs/2101.02636 (2021). arXiv:2101.02636 https://arxiv.org/abs/2101.02636

[183] Pascal J. Sager, Benjamin Meyer, Peng Yan, Rebekka von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F. Grewe, and Thilo Stadelmann. 2025. A Comprehensive Survey of Agents for Computer Use: Foundations, Challenges, and Future Directions. arXiv:2501.16150 [cs.AI] https://arxiv.org/abs/2501.16150

[184] Saeed Salehi, Iman Miremadi, Mobin Ghasempour Nejati, and Hosein Ghafouri. 2024. Fostering the Adoption and Use of Super App Technology. *IEEE Trans. Engineering Management* 71 (2024), 4761–4775. doi:10.1109/TEM.2023.3235718

[185] Jordan Samhi, Jun Gao, Nadia Daoudi, Pierre Graux, Henri Hoyez, Xiaoyu Sun, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2021. JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis. *CoRR*

abs/2112.10469 (2021). arXiv:2112.10469 https://arxiv.org/abs/2112.10469

[186] Feng Shen, Justin Del Vecchio, Aziz Mohaisen, Steven Y. Ko, and Lukasz Ziarek. 2019. Android Malware Detection Using Complex-Flows. *IEEE Trans. Mob. Comput.* 18, 6 (2019), 1231–1245. doi:10.1109/TMC.2018.2861405

[187] Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2023. "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. *CoRR* abs/2308.03825 (2023). arXiv:2308.03825 doi:10.48550/ARXIV.2308.03825

[188] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau. 2019. MoSSOT: An Automated Blackbox Tester for Single Sign-On Vulnerabilities in Mobile Applications. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang (Eds.). ACM, 269–282. doi:10.1145/3321705.3329801

[189] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, stochastic model-based GUI testing of Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 245–256. doi:10.1145/3106237.3106298

[190] Dewen Suo, Lei Xue, Runze Tan, Weihao Huang, and Guozi Sun. 2024. ARAP: Demystifying Anti Runtime Analysis Code in Android Apps. *CoRR* abs/2408.11080 (2024). arXiv:2408.11080 doi:10.48550/ARXIV.2408.11080

[191] Xuchen Suo. 2024. Signed-Prompt: A New Approach to Prevent Prompt Injection Attacks Against LLM-Integrated Applications. *CoRR* abs/2401.07612 (2024). arXiv:2401.07612 doi:10.48550/ARXIV.2401.07612

[192] Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro. 2015. Copperdroid: Automatic reconstruction of android malware behaviors.. In *Ndss*. 1–15.

[193] Hanzhuo Tan, Qi Luo, Jing Li, and Yuqun Zhang. 2024. LLM4Decompile: Decompiling Binary Code with Large Language Models. *CoRR* abs/2403.05286 (2024). arXiv:2403.05286 doi:10.48550/ARXIV.2403.05286

[194] Tian Tan and Yue Li. 2023. Tai-e: A Developer-Friendly Static Analysis Framework for Java by Harnessing the Good Designs of Classics. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, René Just and Gordon Fraser (Eds.). ACM, 1093–1105. doi:10.1145/3597926.3598120

[195] Tencent. 2025. Tinker - Android Hot - fix Solution Library. https://github.com/Tencent/tinker. Accessed: 2025-02-21.

[196] Sercan Türker and Ahmet Burak Can. 2019. AndMFC: Android Malware Family Classification Framework. In *30th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications Workshops, PIMRC Workshops 2019, Istanbul, Turkey, September 8, 2019*. IEEE, 1–6. doi:10.1109/PIMRCW.2019.8880840

[197] Timothy Vidas, Jiaqi Tan, Jay Nahata, Chaur Lih Tan, Nicolas Christin, and Patrick Tague. 2014. A5: Automated Analysis of Adversarial Android Applications. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, SPSM at CCS 2014, Scottsdale, AZ, USA, November 03 - 07, 2014*, Cliff Wang, Dijiang Huang, Kapil Singh, and Zhenkai Liang (Eds.). ACM, 39–50. doi:10.1145/2666620.2666630

[198] Chenxu Wang, Tianming Liu, Yanjie Zhao, Minghui Yang, and Haoyu Wang. 2025. LLMDroid: Enhancing Automated Mobile App GUI Testing Coverage with Large Language Model Guidance. *Proc. ACM Softw. Eng.* 2, FSE (2025), 1001–1022. doi:10.1145/3715763

[199] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. One Size Does Not Fit All: Uncovering and Exploiting Cross Platform Discrepant APIs in WeChat. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 6629–6646. https://www.usenix.org/conference/usenixsecurity23/presentation/wang-chao

[200] Chao Wang, Yanjie Zhao, Jiapeng Deng, and Haoyu Wang. 2025. Born with a Silver Spoon: On the (In)Security of Native Granted App Privileges in Custom Android ROMs. In *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, Marina Blanton, William Enck, and Cristina Nita-Rotaru (Eds.). IEEE, 4267–4283. doi:10.1109/SP61157.2025.00017

[201] Dingbang Wang, Yu Zhao, Sidong Feng, Zhaoxu Zhang, William G. J. Halfond, Chunyang Chen, Xiaoxia Sun, Jiangfan Shi, and Tingting Yu. 2024. Feedback-Driven Automated Whole Bug Report Reproduction for Android Apps. *CoRR* abs/2407.05165 (2024). arXiv:2407.05165 doi:10.48550/ARXIV.2407.05165

[202] Haoran Wang, Xiongxiao Xu, Baixiang Huang, and Kai Shu. 2025. Privacy-Aware Decoding: Mitigating Privacy Leakage of Large Language Models in Retrieval-Augmented Generation. *CoRR* abs/2508.03098 (2025). arXiv:2508.03098 doi:10.48550/ARXIV.2508.03098

[203] Jikai Wang and Haoyu Wang. 2024. NativeSummary: Summarizing Native Binary Code for Inter-language Static Analysis of Android Apps. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, Maria Christakis and Michael Pradel (Eds.). ACM, 971–982. doi:10.1145/3650212.3680335

[204] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-Agent-v2: Mobile Device Operation Assistant with Effective Navigation via Multi-Agent Collaboration. *CoRR* abs/2406.01014 (2024). arXiv:2406.01014 doi:10.48550/ARXIV.2406.01014

[205] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-Agent: Autonomous Multi-Modal Mobile Device Agent with Visual Perception. *CoRR* abs/2401.16158 (2024). arXiv:2401.16158 doi:10.48550/ARXIV.2401.16158

[206] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying Android Malware in the New Era. In *SIGMETRICS/PERFORMANCE '22: ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, Mumbai, India, June 6 - 10, 2022*, D. Manjunath, Jayakrishnan Nair, Niklas Carlsson, Edith Cohen, and Philippe Robert (Eds.). ACM, 21–22. doi:10.1145/3489048.3530973

[207] Pei Wang, Qinkun Bao, Li Wang, Shuai Wang, Zhaofeng Chen, Tao Wei, and Dinghao Wu. 2018. Software protection on the go: a large-scale empirical study on mobile app obfuscation. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 26–36. doi:10.1145/3180155.3180169

[208] Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. 2024. GUI Agents with Foundation Models: A Comprehensive Survey. *CoRR* abs/2411.04890 (2024). arXiv:2411.04890 doi:10.48550/ARXIV.2411.04890

[209] Wenyu Wang, Wing Lam, and Tao Xie. 2021. An infrastructure approach to improving effectiveness of Android UI testing tools. In *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, Cristian Cadar and Xiangyu Zhang (Eds.). ACM, 165–176. doi:10.1145/3460319.3464828

[210] Yin Wang, Ming Fan, Junfeng Liu, Junjie Tao, Wuxia Jin, Qi Xiong, Yuhao Liu, Qinghua Zheng, and Ting Liu. 2023. Do as You Say: Consistency Detection of Data Practice in Program Code and Privacy Policy in Mini-App. *CoRR* abs/2302.13860 (2023). arXiv:2302.13860 doi:10.48550/ARXIV.2302.13860

[211] Yingjie Wang, Guangquan Xu, Xing Liu, Weixuan Mao, Chengxiang Si, Witold Pedrycz, and Wei Wang. 2020. Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis. *J. Syst. Softw.* 167 (2020), 110609. doi:10.1016/J.JSS.2020.110609

[212] Zhitao Wang, Wei Wang, Zirao Li, Long Wang, Can Yi, Xinjie Xu, Luyang Cao, Hanjing Su, Shouzhi Chen, and Jun Zhou. 2024. XUAT-Copilot: Multi-Agent Collaborative System for Automated User Acceptance Testing with Large Language Model. *CoRR* abs/2401.02705 (2024). arXiv:2401.02705 doi:10.48550/ARXIV.2401.02705

[213] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025. Mobile-Agent-E: Self-Evolving Mobile Assistant for Complex Tasks. *CoRR* abs/2501.11733 (2025). arXiv:2501.11733 doi:10.48550/ARXIV.2501.11733

[214] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. 2014. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 1329–1341. doi:10.1145/2660267.2660357

[215] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. AutoDroid: LLM-powered Task Automation in Android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, ACM MobiCom 2024, Washington D.C., DC, USA, November 18-22, 2024*, Weisong Shi, Deepak Ganesan, and Nicholas D. Lane (Eds.). ACM, 543–557. doi:10.1145/3636534.3649379

[216] Daoyuan Wu, Debin Gao, Rocky K. C. Chang, En He, Eric K. T. Cheng, and Robert H. Deng. 2019. Understanding Open Ports in Android Applications: Discovery, Diagnosis, and Security Assessment. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/understanding-open-ports-in-android-applications-discovery-diagnosis-and-security-assessment/

[217] Fangzhou Wu, Ethan Cecchetti, and Chaowei Xiao. 2024. System-Level Defense against Indirect Prompt Injection Attacks: An Information Flow Control Perspective. *CoRR* abs/2409.19091 (2024). arXiv:2409.19091 doi:10.48550/ARXIV.2409.19091

[218] Liangxuan Wu, Chao Wang, Tianming Liu, Yanjie Zhao, and Haoyu Wang. 2025. From Assistants to Adversaries: Exploring the Security Risks of Mobile LLM Agents. *CoRR* abs/2505.12981 (2025). arXiv:2505.12981 doi:10.48550/ARXIV.2505.12981

[219] Liangxuan Wu, Yanjie Zhao, Chao Wang, Tianming Liu, and Haoyu Wang. 2024. A First Look at LLM-powered Smartphones. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops* (Sacramento, CA, USA) *(ASEW '24)*. Association for Computing Machinery, New York, NY, USA, 208–217. doi:10.1145/3691621.3694952

[220] Danning Xie, Zhuo Zhang, Nan Jiang, Xiangzhe Xu, Lin Tan, and Xiangyu Zhang. 2024. ReSym: Harnessing LLMs to Recover Variable and Data Structure Symbols from Stripped Binaries. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 4554–4568. doi:10.1145/3658644.3670340

[221] Chejian Xu, Mintong Kang, Jiawei Zhang, Zeyi Liao, Lingbo Mo, Mengqi Yuan, Huan Sun, and Bo Li. 2024. AdvWeb: Controllable Black-box Attacks on VLM-powered Web Agents. *CoRR* abs/2410.17401 (2024). arXiv:2410.17401 doi:10.48550/ARXIV.2410.17401

[222] Ke Xu, Yingjiu Li, Robert H. Deng, and Kai Chen. 2018. DeepRefiner: Multi-layer Android Malware Detection System Applying Deep Neural Networks. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 473–487. doi:10.1109/EUROSP.2018.00040

[223] Minrui Xu, Jiani Fan, Xinyu Huang, Conghao Zhou, Jiawen Kang, Dusit Niyato, Shiwen Mao, Zhu Han, Xuemin Shen, and Kwok-Yan Lam. 2025. Forewarned is Forearmed: A Survey on Large Language Model-based Agents in Autonomous Cyberattacks. *CoRR* abs/2505.12786 (2025). arXiv:2505.12786 doi:10.48550/ARXIV.2505.12786

[224] Zhiwu Xu, Kerong Ren, and Fu Song. 2019. Android Malware Family Classification and Characterization Using CFG and DFG. In *2019 International Symposium on Theoretical Aspects of Software Engineering, TASE 2019, Guilin, China, July 29-31, 2019*, Dominique Méry and Shengchao Qin (Eds.). IEEE, 49–56. doi:10.1109/TASE.2019.00-20

[225] Chuan Yan, Mark Huasong Meng, Fuman Xie, and Guangdong Bai. 2024. Investigating Documented Privacy Changes in Android OS. *Proc. ACM Softw. Eng.* 1, FSE (2024), 2701–2724. doi:10.1145/3660826

[226] Wenkai Yang, Xiaohan Bi, Yankai Lin, Sishuo Chen, Jie Zhou, and Xu Sun. 2024. Watch Out for Your Agents! Investigating Backdoor Threats to LLM-Based Agents. *CoRR* abs/2402.11208 (2024). arXiv:2402.11208 doi:10.48550/ARXIV.2402.11208

[227] Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. 2015. AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, 303–313. doi:10.1109/ICSE.2015.50

[228] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2022. Cross Miniapp Request Forgery: Root Causes, Attacks, and Vulnerability Detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 3079–3092. doi:10.1145/3548606.3560597

[229] Zhi Yang, Zhanhui Yuan, Shuyuan Jin, Xingyuan Chen, Lei Sun, Xuehui Du, Wenfa Li, and Hongqi Zhang. 2022. FSAFlow: Lightweight and Fast Dynamic Path Tracking and Control for Privacy Protection on Android Using Hybrid Analysis with State-Reduction Strategy. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2114–2129. doi:10.1109/SP46214.2022.9833764

[230] Yifan Yao, Shawn McCollum, Zhibo Sun, and Yue Zhang. 2025. Easy As Child's Play: An Empirical Study on Age Verification of Adult-Oriented Android Apps. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, Lujo Bauer and Giancarlo Pellegrino (Eds.). USENIX Association, 21–39. https://www.usenix.org/conference/usenixsecurity25/presentation/yao-yifan

[231] Alperen Yildiz, Sin G. Teo, Yiling Lou, Yebo Feng, Chong Wang, and Dinil Mon Divakaran. 2025. Benchmarking LLMs and LLM-based Agents in Practical Vulnerability Detection for Code Repositories. *CoRR* abs/2503.03586 (2025). arXiv:2503.03586 doi:10.48550/ARXIV.2503.03586

[232] Juyeon Yoon, Robert Feldt, and Shin Yoo. 2024. Intent-Driven Mobile GUI Testing with Autonomous Large Language Model Agents. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 129–139. doi:10.1109/ICST60714.2024.00020

[233] Ting Yuan, Wenrui Zhang, Dong Chen, and Jie Wang. 2025. CG-Bench: Can Language Models Assist Call Graph Construction in the Real World?. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Language Models and Programming Languages* (Singapore, Singapore) *(LMPL '25)*. Association for Computing Machinery, New York, NY, USA, 12–20. doi:10.1145/3759425.3763379

[234] Chang Yue, Kai Chen, Zhixiu Guo, Jun Dai, Xiaoyan Sun, and Yi Yang. 2025. What's Done Is Not What's Claimed: Detecting and Interpreting Inconsistencies in App Behaviors. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/whats-done-is-not-whats-claimed-detecting-and-interpreting-inconsistencies-in-app-behaviors/

[235] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. AppAgent: Multimodal Agents as Smartphone Users. *CoRR* abs/2312.13771 (2023). arXiv:2312.13771 doi:10.48550/ARXIV.2312.13771

[236] Guangyu Zhang, Xixuan Wang, Shiyu Sun, Peiyan Xiao, Kun Sun, and Yanhai Xiong. 2025. TraceRAG: A LLM-Based Framework for Explainable Android Malware Detection and Behavior Analysis. *CoRR* abs/2509.08865 (2025).

arXiv:2509.08865 doi:10.48550/ARXIV.2509.08865

[237] Hanqing Zhang, Senlin Luo, Yifei Zhang, and Limin Pan. 2019. An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis. *IEEE Access* 7 (2019), 69246–69256. doi:10.1109/ACCESS.2019.2919796

[238] Mu Zhang and Heng Yin. 2014. Efficient, context-aware privacy leakage confinement for android applications without firmware modding. In *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, Shiho Moriai, Trent Jaeger, and Kouichi Sakurai (Eds.). ACM, 259–270. doi:10.1145/2590296.2590312

[239] Weizhe Zhang, Huanran Wang, Hui He, and Peng Liu. 2020. DAMBA: Detecting Android Malware by ORGB Analysis. *IEEE Trans. Reliab.* 69, 1 (2020), 55–69. doi:10.1109/TR.2019.2924677

[240] Yuyang Zhang, Kangjie Chen, Xudong Jiang, Yuxiang Sun, Run Wang, and Lina Wang. 2024. Towards Action Hijacking of Large Language Model-based Agent. *CoRR* abs/2412.10807 (2024). arXiv:2412.10807 doi:10.48550/ARXIV.2412.10807

[241] Yuxin Zhang, Sen Chen, Xiaofei Xie, Zibo Liu, and Lingling Fan. 2025. Scenario-Driven and Context-Aware Automated Accessibility Testing for Android Apps. In *47th IEEE/ACM International Conference on Software Engineering, ICSE 2025, Ottawa, ON, Canada, April 26 - May 6, 2025*. IEEE, 2777–2789. doi:10.1109/ICSE55347.2025.00093

[242] Yakun Zhang, Chen Liu, Xiaofei Xie, Yun Lin, Jin Song Dong, Dan Hao, and Lu Zhang. 2024. LLM-based Abstraction and Concretization for GUI Test Migration. *CoRR* abs/2409.05028 (2024). arXiv:2409.05028 doi:10.48550/ARXIV.2409.05028

[243] Wenxiang Zhao, Juntao Wu, and Zhaoyi Meng. 2024. AppPoet: Large Language Model based Android malware detection via multi-view prompt engineering. *CoRR* abs/2404.18816 (2024). arXiv:2404.18816 doi:10.48550/ARXIV.2404.18816

[244] Xin Zhao, Mu Zhang, Xiaopeng Ke, Yu Pan, Yue Duan, Sheng Zhong, and Fengyuan Xu. 2025. DeepVMUnProtect: Neural Network-Based Recovery of VM-Protected Android Apps for Semantics-Aware Malware Detection. *IEEE Trans. Inf. Forensics Secur.* 20 (2025), 3689–3704. doi:10.1109/TIFS.2025.3550049

[245] Yujie Zhao, Zhanyong Tang, Guixin Ye, Dongxu Peng, Dingyi Fang, Xiaojiang Chen, and Zheng Wang. 2020. Compile-time code virtualization for android applications. *Comput. Secur.* 94 (2020), 101821. doi:10.1016/J.COSE.2020.101821

[246] Yijun Zhao, Lingjing Yu, Yong Sun, Qingyun Liu, and Bo Luo. 2024. No Source Code? No Problem! Demystifying and Detecting Mask Apps in iOS. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, ICPC 2024, Lisbon, Portugal, April 15-16, 2024*, Igor Steinmacher, Mario Linares-Vásquez, Kevin Patrick Moran, and Olga Baysal (Eds.). ACM, 358–369. doi:10.1145/3643916.3644419

[247] Rui Zheng, Zhibo Wang, Kui Ren, and Chun Chen. 2025. AV-Agent: A Bottom-Up Interpretable Malware Classifier Based on Large Language Models. *IEEE Trans. Inf. Forensics Secur.* 20 (2025), 8555–8569. doi:10.1109/TIFS.2025.3597221

[248] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. 2024. Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead. *CoRR* abs/2404.02525 (2024). arXiv:2404.02525 doi:10.48550/ARXIV.2404.02525

[249] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R. Reidenberg, N. Cameron Russell, and Norman M. Sadeh. 2019. MAPS: Scaling Privacy Compliance Analysis to a Million Apps. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 66–86. doi:10.2478/POPETS-2019-0037

[250] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. *CoRR* abs/2307.15043 (2023). arXiv:2307.15043 doi:10.48550/ARXIV.2307.15043