

APPBDS: LLM-Powered Description Synthesis for Sensitive Behaviors in Mobile Apps

Zichen Liu

*School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ, USA
zliu396@asu.edu*

Xusheng Xiao

*School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ, USA
xusheng.xiao@asu.edu*

Abstract—As mobile applications (i.e., apps) increasingly manage a wide variety of user needs, their access to sensitive data intensifies privacy concerns among users. While app markets employ permissions to regulate private data access, the lack of explanation for permission usage renders this mechanism less effective. Existing techniques that extract explanatory sentences from app descriptions to inform users about sensitive behaviors are also limited. Many app behaviors remain unexplained in app descriptions. To address these issues, we propose APPBDS, a novel approach that integrates program analysis with Large Language Models (LLMs) to process code semantics and UI contexts, further complemented by privacy policies and information from similar apps, in order to generate detailed explanations for apps' sensitive behaviors. Specifically, APPBDS integrates code semantics with UI contexts to build a UI-Fused Call Graph (UCG) for each app. Additionally, APPBDS summarizes permission-related propositions from privacy policies and utilizes similar apps' information from a knowledge base (PP-KB) to improve LLMs' domain knowledge in explaining permission usage. Our evaluation on 270 real-world apps demonstrates that APPBDS significantly outperforms state-of-the-art approaches in richness, specificity, and semantic relatedness, while also proving highly robust against common code obfuscation.

I. INTRODUCTION

Advancements in smartphone technology have paved the way for a greater variety of mobile applications (apps), attracting an ever-growing number of users. However, aggressive use of users' privacy data by these apps raises serious concerns [1], [2], [3], [4]. To address these concerns, app stores have introduced stringent privacy standards—for instance, the Apple App Store and Google Play employ “nutrition labels” [5], [6] and runtime permission requests [7] to enhance transparency. Despite these measures, research shows that the presented information is often limited and unintuitive [8], [9], [10], as most approaches focus on what data is collected, rather than how it is actually used, leaving users with an incomplete understanding of privacy risks.

To address these limitations, existing research efforts [8], [11] have been put forth to provide descriptions by identifying sentences in app descriptions. As shown in these studies, while app descriptions are mainly designed to describe apps' functionalities and features, they also include sentences that provide justifications for certain permissions. Unfortunately, the studies also reveal that only 37.37% of the sensitive behaviors in the surveyed apps (1,292) have descriptions that explain the apps' sensitive behaviors, and thus these

approaches are not reliable. There also exist code comment generation techniques and permission translation techniques based on predefined models [12], [13], [14], but they mainly focus on explaining the functionality of the code rather than the security implication of the apps' behaviors.

With the rise of neural language translation techniques [15], [16], research has progressed toward automated methods that synthesize natural language explanations to directly describe how users' private data is used. For example, a recent approach [9] trains an encoder-decoder model to translate code semantics, UI texts, and privacy policies into descriptions that explain apps' permission usage. These descriptions can be integrated into runtime permission dialogs to inform users of potential privacy risks and support informed decision-making. However, due to the vast amount of app data and the limited capacity of encoder-decoder models, such approaches struggle to generate detailed and app-specific behavior descriptions, especially for rare behaviors that are underrepresented in the training data and for long input sequences that exceed the model's processing limits.

The rapid evolution of Large Language Models (LLMs) [17], [18], [19], [20], [21], equipped with extensive knowledge and strong reasoning and synthesis capabilities, opens new opportunities for generating high-quality permission usage explanations. In this paper, we present APPBDS, a novel approach that leverages LLMs to process contextual information from code semantics and UI contexts, along with privacy policies and similar apps, to generate rich, app-specific descriptions of sensitive behaviors. We focus on Android apps due to their market dominance and the openness of the development ecosystem. Still, despite the advanced comprehension and synthesis abilities of LLMs, applying them directly to generate descriptions for apps' permission uses poses several major challenges.

First, as shown in the recent study [9], [11], [22], understanding permission usage requires integrating three complementary information sources: code semantics, UI contexts, and privacy policies. Each source provides only a partial view of the app's behavior: code reveals implementation details but lacks explanation, privacy policies offer general descriptions but may be copied from similar apps, and UI contexts provide user-facing information that can bridge the gap between implementation and policy. These three types of information

are distributed differently and contain distinct information for behavior explanation, making it difficult to generate high-quality descriptions based on a single source.

Second, handling the complex structure within each information source poses another significant challenge. App code contains extensive implementation details, with only a small portion relevant to permission usage. Similarly, UI contexts include layout information and resources that may not contribute to understanding sensitive behaviors. Although LLMs can handle vast inputs, the presence of irrelevant information significantly hinders their performance, making effective filtering and prioritization essential yet technically challenging.

Third, privacy policies are notoriously lengthy, difficult to understand, and often of questionable quality [9], [22]. They often contain a substantial amount of irrelevant information, such as legal information and user agreements. Furthermore, many privacy policies serve multiple apps from the same developer, containing generic descriptions rather than app-specific explanations. Since privacy policies provide the primary vocabulary for permission descriptions, low-quality policies tend to produce generic explanations like “Microphone is used to record voice” instead of specific descriptions such as “Use the microphone for real-time repetition of your speech with fun voice effects.”

To address these challenges, the key insight of APPBDS lies in the innovative integration of program analysis and LLMs. APPBDS systematically identifies and summarizes privacy-related code elements and UI contexts, incorporates behavioral knowledge from similar apps, and guides LLMs to extract concise, privacy-relevant propositions from complex privacy policies. This approach filters out irrelevant information while preserving critical insights into app behaviors.

Specifically, APPBDS is powered by the following novel designs.

- ① *UI-Fused Call Graph (UCG) Construction:* we propose a novel technique that integrates apps’ UI contexts with code structures and semantics and represents them as a UCG. In particular, APPBDS extends static call graph analysis with Android framework APIs to build the Inter-Component Call Graph (ICCG) of apps [3], [2], [23], [24], extracts texts from UI layout files and decompiled code from the APK file, and integrates them to construct the UCG of an app.
- ② *Privacy-Relevant Behavior Summarization:* An UCG usually contains many irrelevant ones (e.g., obfuscated internal functions and system utility methods), introducing noise that can trigger hallucinations in subsequent LLM analysis. Thus, APPBDS employs node patterns summarized from prior research [24], [25], [3], [2], [26] and embedding-similarity-based retrieval methods to identify privacy-relevant nodes that preserves privacy-critical behavior information while effectively minimizing noise. To facilitate LLMs in analyzing these privacy-relevant nodes, APPBDS further employs LLMs to summarize the UI contexts and their associated code behaviors described in these nodes, facilitating LLMs in consuming these two types of originally disconnected information.

- ③ *Permission Proposition Inference:* To consume information from privacy policies, APPBDS employs LLMs to filter out irrelevant information and extract essential descriptions for declared permissions. Specifically, APPBDS instructs LLMs to process privacy policies and summarize key points and claims as *permission-related propositions*, eliminating the need to process lengthy privacy policy documents (referred to as *PP*) in later steps.

- ④ *Multi-Agent:* To address the low quality of some privacy policies, we observe that similar apps can help improve weak permission descriptions. APPBDS constructs a Permission-related Proposition Knowledge Base (PP-KB) by applying LLMs to extract permission-related propositions from a large set of apps. By measuring the similarity between an app’s propositions and those in the knowledge base, APPBDS supplements an app’s own propositions with those of highly similar apps, guiding LLMs to analyze the UCG for supporting evidence.

Through these innovative designs, APPBDS can seamlessly process heterogeneous information extracted from code, UI contexts, and privacy policies, effectively filtering out noisy data to produce detailed, app-specific descriptions of permission usage. Our evaluations on 270 real apps show that APPBDS greatly outperforms the state-of-the-art approach, DescribeCTX [9], for the semantic metrics and the automatic LLM evaluation metrics [27], [28], [29] in terms of factuality and semantic richness.

This paper makes the following contributions:

- A novel LLM-based approach for automatically generating app permission usage descriptions by integrating code semantics, UI contexts, and privacy policies.
- An advanced method for constructing UCGs by integrating UI and code semantics.
- A knowledge retrieval technique for leveraging similar apps to expand LLMs’ domain knowledge.
- Comprehensive evaluation and ablation studies showing the superiority of APPBDS over the state-of-the-art, with improvements of 729%, 517%, and 656% in richness, specificity, and semantic relatedness, respectively.
- An empirical validation of the framework’s practical robustness and generalizability, demonstrating high performance against common code obfuscation and across multiple open-source LLMs.

Our project is available at our project website [30].

II. OVERVIEW

As shown in Figure 1, APPBDS consists of three phases: Multi-source Data Acquisition and Preparation (Phase I), Privacy-Relevant Node Filter and Summarization (Phase II), and Permission Description Synthesis (Phase III).

Phase I prepares three heterogeneous types of information. First, APPBDS performs static analysis on the input APK to extract GUI layouts containing UI context information and the ICCG, then integrates these elements by mapping UI contextual resources from the GUI layouts to the ICCG nodes to create a “Ⓐ UCG”. Second, APPBDS processes the app’s PP and a sensitive permission from the app’s metadata,

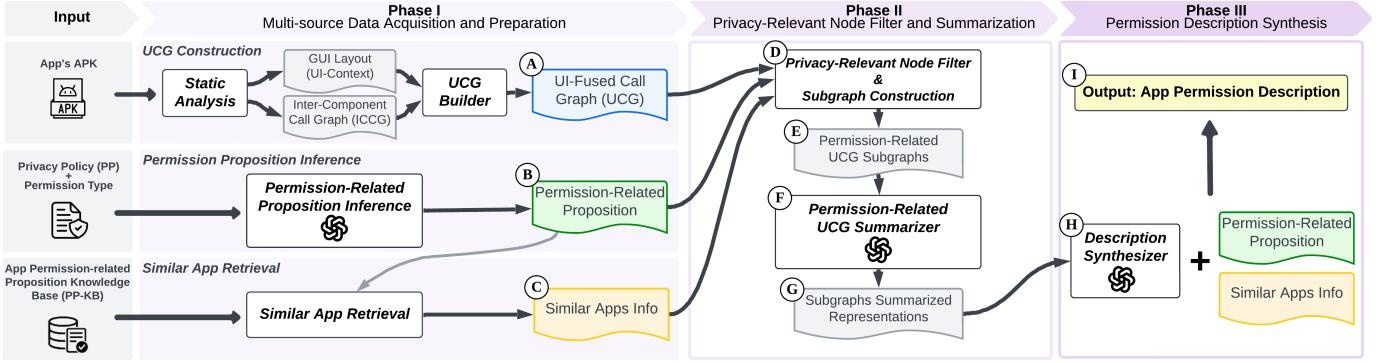


Fig. 1: Overview of the APPBDS framework

employing an LLM agent to generate “(B) Permission-Related Propositions”. Third, APPBDS queries the PP-KB to find similar apps from the PP-KB based on proposition similarity and retrieves their associated information, forming “(C) Similar App Info”.

Phase II takes the output of Phase I (A, B, C) as input, and generates a summary of the relevant information (“(G) Subgraphs Summarized Representations”). Specifically, APPBDS implements a “(D) Privacy-Relevant Node Filter” that leverages multiple predefined critical node patterns to extract nodes containing critical privacy-related information from the UCG. These extracted nodes are then reconstructed into three “(E) Permission-Related UCG Subgraphs”. APPBDS then employs an LLM agent (“(F) Permission-Related UCG Summarizer”) to summarize each node within these subgraphs to generate the summaries.

Phase III implements a “(H) Description Synthesizer” that employs LLM agents to process the summaries of the three subgraphs, and perform cross-validation and integration of the heterogeneous analysis results to synthesize the “(I) App Permission Description”.

A. Example Workflow of APPBDS

Figure 2 provides a step-by-step illustration of APPBDS’s workflow using a simplified real-world app (`com.gec.MarineApp`), including examples of the input sources, intermediate steps, and the generated description.

In Phase I, (A) depicts a node from the constructed UCG, showing the method signature, a function body that defines three alert dialog types (*poor GPS*, *low battery*, and *outside allowed distance*), and a UI context featuring an *anchor alarm* capability. (B) depicts PP segments related to location data, alongside two inferred permission-related propositions: *Proposition 1* specifies continuous location tracking for real-time navigation, and *Proposition 2* identifies location sharing with support staff for safety purposes. (C) presents retrieved information of a similar app from the PP-KB, including a relevant proposition and a description emphasizing security and safety purposes for location data usage.

In Phase II, (D) shows the Privacy-Relevant Node Filter, which identifies relevant nodes using six patterns. (E) illustrates the construction of three subgraphs from these nodes: a general subgraph, a proposition-specialized subgraph, and a

similar-apps-specialized subgraph. (F) shows the Permission-Related UCG Summarizer, where LLM agents use specialized prompts tailored to different node types to generate summarized representations for each subgraph. (G) shows an example of such a summarized representation for the `onCreate()` method of the class `ShowAnchorAlertDialog`.

In Phase III, (H) shows the Description Synthesizer. This module includes three LLM agents: one extracts permission-related app features (Permission-Focused Analysis), another matches code implementation with policy claims (Policy Proposition Validation), and the third infers related features from similar apps (Similar-App-Inspired Discovery). An aggregator agent then integrates the findings from all three agents.

The final output, (I), shows the generated descriptions for two permission-related features. The *anchor mark and distance alert* functionality is synthesized from the function code and UI evidence in (A), further informed by the safety-focused usage patterns from the similar app in (C). The *location tracking* feature is derived from a separate `TrackInfoActivity` node (not shown in this example node (A)), whose UI context includes speed and altitude indicators, consistent with Proposition 1 from (B). Proposition 2 from (B) (location sharing with support staff) is excluded from the final description due to lack of supporting evidence in the UCG analysis.

III. APPROACH

A. Phase I: Multi-source Data Acquisition and Preparation

UCG Construction. UCG Construction includes two components (Static Analysis and UCG Builder) that extract code semantics and UI contexts from the input APK to create a UCG (A).

Static Analysis. This component constructs an ICCG, which is essential for analyzing Android apps as they are fundamentally built upon app components [24], [2], [9]. An ICCG is a directed call graph where nodes represent methods with their signatures and decompiled method bodies, and edges represent calling relationships between methods. To construct the ICCG, we leverage the Soot [31] and FlowDroid [32] frameworks to build a static call graph enhanced with Android-specific implicit calling relationships, including multi-threading, event-driven, and inter-component communication (ICC) methods [23], [33]. Although an ICCG captures calling relationships and code logic, it lacks essential semantic information such

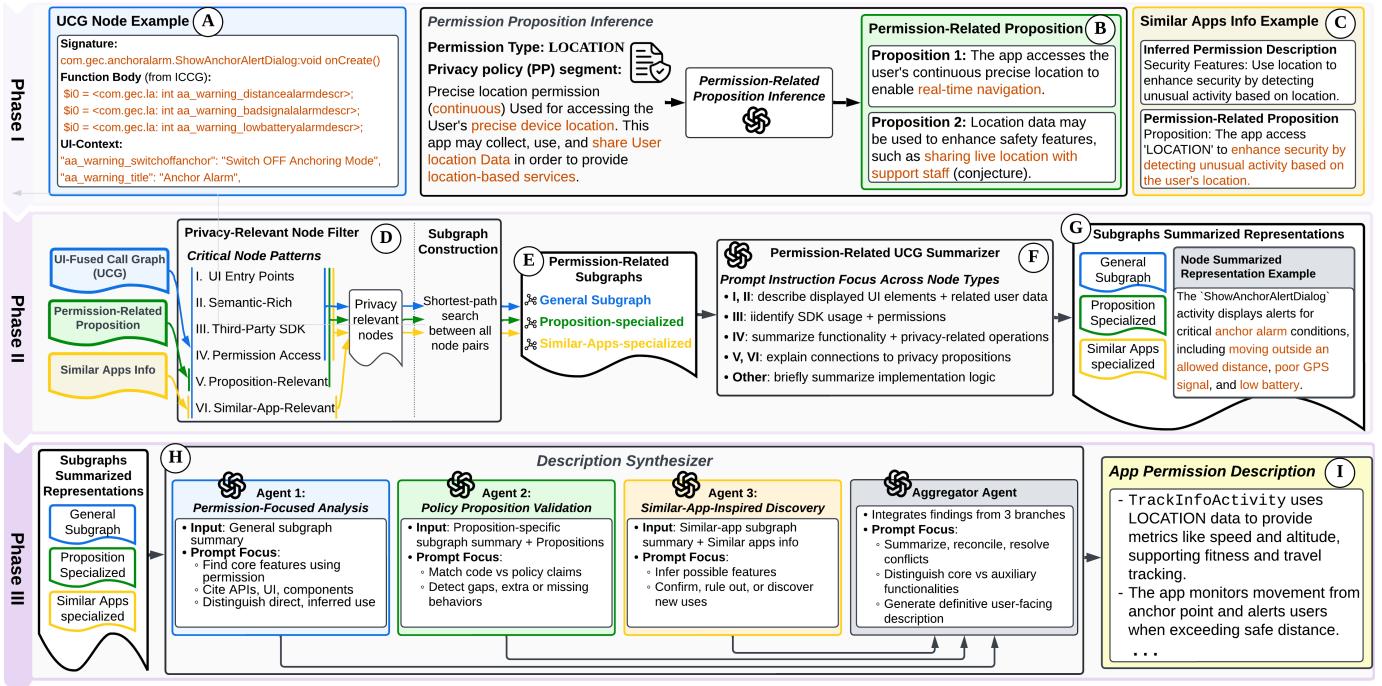


Fig. 2: Example workflow of APPBDS on a real-world app com.gec.MarineApp

as UI contexts that would make the app's functionality comprehensible to users. Therefore, APPBDS incorporates textual information from GUI layout XML files to provide descriptive insights into the app's behaviors and their UI contexts.

UCG Builder. This component enriches the ICCG with UI contexts by analyzing both activity-specific and general node information. For activity nodes, which serve as fundamental components for user interfaces, this component parses the `AndroidManifest.xml` to extract activities and identifies their associated UI layout files. For all nodes in the ICCG, this component analyzes method bodies to extract UI-related operations and resource references, using pattern matching to identify layout resource identifiers, string resources, and hardcoded constants. The `public.xml` resource mapping file is utilized to resolve resource identifiers to their actual definitions, establishing connections between code operations and UI elements. The resulting UCG extends the original ICCG by associating each node with relevant UI context information, providing a comprehensive view of how code logic and UI elements interact, thus enabling downstream analysis of both calling relationships and UI-driven behaviors.

Permission Proposition Inference. Since privacy policies often contain irrelevant or potentially inaccurate information [9], [22], APPBDS employs an LLM-based analysis to identify permission-relevant segments within the policy and synthesize them into structured Permission-Related Propositions (B). These propositions capture critical information and possible usage scenarios for the target permission.

Each proposition serves as a directional statement that can be independently verified against code evidence and similar app behaviors. This proposition-based approach offers several advantages: it distills complex and often verbose policy statements into concise, analyzable claims; it allows for reasonable

inference beyond literal policy text to address incomplete descriptions; and it provides specific, testable hypotheses that guide subsequent evidence collection and validation in Phase II's Privacy-Relevant Node Filter and Phase III's Policy Proposition Validation during description synthesis. Additionally, these propositions facilitate similar app retrieval from the PP-KB by enabling semantic comparison of permission usage patterns.

Similar App Retrieval. While Permission Proposition Inference extracts propositions from the target app's PP, these propositions may be limited by incomplete or low-quality policy descriptions. To address this limitation, similar app retrieval leverages information from similar apps to provide diverse perspectives and potential functionality insights that may not be explicitly documented in the target app's PP.

PP-KB Construction. We collect thousands of popular apps from Google Play [34] to construct a PP-KB. For each app in the PP-KB, APPBDS performs a two-step process: first, it extracts permission-related propositions using the same analysis pipeline as Permission Proposition Inference; second, it generates inferred permission descriptions by leveraging web search capabilities to gather app-related documentation, technical blogs, and functionality descriptions, then uses this collected information to provide answers to the generated propositions. This approach enables the creation of diverse permission descriptions.

Similar App Retrieval Process. APPBDS identifies similar apps from the PP-KB by computing cosine similarities between proposition embeddings. For each target app, APPBDS computes embeddings for its propositions and compares them with propositions from PP-KB apps sharing the same permission type. The overall similarity between apps is calculated by averaging all proposition pair similarities, and the top- k most

similar apps are selected.

The retrieved similar apps contribute primarily through functional diversity rather than strict similarity, providing alternative implementation patterns, potential use cases, and permission usage scenarios that inspire the identification of implicit or undocumented functionalities in the target app. This approach enables subsequent description synthesis to explore broader functionality possibilities beyond the limitations of the target app's PP, prioritizing coverage and insight generation over strict accuracy validation.

B. Phase II: Privacy-Relevant Node Filter and Summarization

This phase employs a Privacy-Relevant Node Filter to identify essential UCG nodes using six distinct patterns, constructs three specialized subgraphs that capture different analytical perspectives, and generates summarized representations to facilitate subsequent LLM-based analysis.

Privacy-Relevant Node Filter. As shown in Figure 2 (D), APPBDS identifies privacy-relevant nodes from UCG using six node patterns, each designed to capture specific types of evidence crucial for permission-related behavior analysis:

Pattern I - UI Entry Points: UCG nodes representing activity classes are crucial because they serve as user interaction entry points and contain rich UI semantics and business logic. APPBDS locates these nodes by detecting activity-related signatures and employs LLMs to determine their permission relevance, prioritizing `onCreate()` methods that initialize UI components and house core functionality.

Pattern II - Semantic-Rich Implementation: UCG nodes associated with semantic resources (e.g., layouts, strings) often represent dialogs or UI fragments. APPBDS assesses semantic richness by examining resource references and calculating weighted scores based on the diversity and specificity of UI-related content.

Pattern III - Third-Party SDK: Some UCG nodes utilize third-party SDKs for functionality like advertising, analytics, and social sharing. They provide valuable information due to consistent naming patterns and comprehensive documentation. APPBDS identifies these nodes using a curated mapping table based on the Google Play SDK Index [35].

Pattern IV - Permission Access: APPBDS maintains a comprehensive list of permission-protected APIs to identify method calls of permission-protected APIs within the UCG.

Pattern V - Proposition-Relevant: APPBDS uses embedding-based similarity retrieval to identify nodes whose functionality aligns with the permission-related propositions extracted from the app's PP, thereby enabling evidence-based verification of policy statements.

Pattern VI - Similar-App-Relevant: APPBDS identifies nodes that implement functionalities observed in similar apps, leveraging this cross-app comparison to uncover potential undocumented features or alternative implementation strategies.

Subgraph Construction. To facilitate downstream analysis, APPBDS constructs three specialized subgraphs from the identified privacy-relevant nodes (see Figure 2 (D, E)):

The *General Subgraph* contains nodes from Patterns I-IV, providing a comprehensive view of core permission-related

functionality independent of specific propositions or similar app influences.

The *Proposition-Specialized Subgraph* extends the general subgraph by incorporating Pattern V nodes, creating a targeted representation for verifying specific claims made in the target app's PP.

The *Similar-Apps-Specialized Subgraph* extends the general subgraph with Pattern VI nodes, enabling exploration of potential functionalities inspired by similar app behaviors.

For each subgraph, APPBDS expands the identified nodes into complete subgraph structures by employing a shortest-path-based algorithm that preserves calling relationships. This process identifies all paths between critical nodes using Breadth-First Search (BFS), incorporates intermediate nodes along these paths, and maintains original calling relationships as edges, ensuring that the execution context and dependency relationships are preserved for accurate behavioral understanding.

Permission-Related UCG Summarizer (F). While subgraphs effectively capture privacy-relevant calling relationships, they must be converted to text formats for LLM-based analysis, given the limitations in handling complex graph structures [36], [37]. APPBDS employs LLMs to summarize each node with specialized prompts tailored to different node types.

For UI-related nodes (Patterns I-II), the prompt emphasizes extracting user-facing functionality descriptions, interface elements, and user interaction workflows. For SDK nodes (Pattern III), the focus shifts to identifying specific third-party service integrations and their privacy implications. For permission access nodes (Pattern IV), the summarization concentrates on direct API usage patterns and data access behaviors. For proposition and similar-app relevant nodes (Patterns V-VI), the prompts guide the analysis to explain connections to privacy propositions and similar-app functionalities.

The resulting node summaries are concatenated to form the summarized representation of each subgraph, providing structured input for the subsequent description synthesis phase.

C. Phase III: Permission Description Synthesis

Phase III employs a parallel analysis architecture to generate comprehensive permission descriptions. As illustrated in Figure 2 (H), the Description Synthesizer consists of three specialized LLM agents followed by an aggregator agent.

- **Agent 1 - Permission-Focused Analysis:** This agent processes the general subgraph summary to identify core permission functionality patterns, focusing on distinguishing direct versus inferred usage and determining primary functional purposes.
- **Agent 2 - Policy Proposition Validation:** This agent combines the proposition-specialized subgraph summary with permission-related propositions to evaluate policy accuracy, focusing on matching code behaviors against policy claims and detecting gaps or inconsistencies.
- **Agent 3 - Similar-App-Inspired Discovery:** This agent processes the similar-apps-specialized subgraph summary alongside similar apps information to explore potential

undocumented functionalities, focusing on inferring possible features and discovering alternative implementation approaches.

- **Aggregator Agent:** This agent integrates the findings from all three agents, reconciling conflicts and identifying consistent patterns to produce a coherent, evidence-grounded description capturing both explicit and implicit permission usage behaviors.

IV. EVALUATION

We evaluate the effectiveness of APPBDS on real-world apps. Specifically, we aim to answer the following research questions:

- **RQ1:** How effective is APPBDS in generating descriptions for apps' sensitive behaviors? How does APPBDS compare with the existing work?
- **RQ2:** How well do the LLM-based evaluation metrics align with human perception?
- **RQ3:** How effective are the processing mechanisms in each phase of APPBDS?
- **RQ4:** How robust is APPBDS's performance against code obfuscation?
- **RQ5:** What are the common errors in the generated descriptions and what causes them?

A. Evaluation Setup

Evaluation Subject. We focus on seven key types of private information that are protected by dangerous permissions in Android system: CALENDAR, CAMERA, CONTACT, LOCATION, MICROPHONE, SMS, and STORAGE. These dangerous permission types all relate to user-sensitive data and represent the main categories that protect user-understandable sensitive information [38], [39], [11], [40], [41]. We collected real-world apps from Google Play [34] and AndroZoo [42] to form our evaluation dataset. The criteria for selecting apps for the evaluation dataset are as follows:

- **Permission:** We excluded apps if their requested permissions do not belong to our focused permissions. This information is derived from the app's manifest file.
- **Privacy Policy:** As privacy policies are part of APPBDS's inputs and are required for manual inspection of privacy usage, we excluded apps (1) that do not have privacy policies or (2) whose privacy policies do not contain descriptions for the requested permissions.
- **App Description:** As shown in prior research [8], [9], app descriptions contain sentences that explain permission uses, which is critical for our manual inspection of apps' privacy usage. Thus, we excluded apps if their app descriptions do not contain sentences that explain the uses of the requested permissions.

In total, we selected 270 apps for our evaluation, and data distribution based on the types of requested permissions is shown in Table I. The selected apps cover a wide range of popularity, from apps with few downloads to top-downloaded apps (Figure 3), ensuring that the dataset is representative and reflective of real-world market distribution. For each app, we also assign a PP quality rating (0–5) to indicate whether the

TABLE I: Test set and PP-KB data distribution

Permission	Test Set	PP-KB
CALENDAR	6	500
CAMERA	69	500
CONTACT	11	500
LOCATION	73	500
MICROPHONE	30	500
SMS	2	500
STORAGE	79	500
Total	270	3500

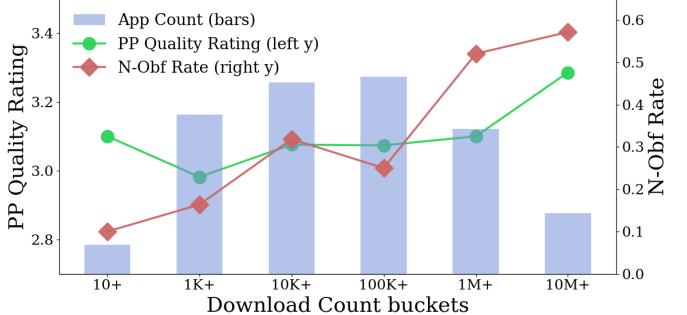


Fig. 3: Privacy-policy quality and native obfuscation across download count buckets. Bars show app counts in each bucket; the green line (circles, left y) is the average PP quality rating (0–5); the red line (diamonds, right y) is the native-obfuscation (N-Obf) rate.

policy explicitly discloses the focused sensitive permission and its concrete usage. The results show that many apps provide low-quality or even missing PP. This motivates APPBDS's design to leverage PP from apps with similar functionality to enhance explanations of permission usage, as apps with similar functionality and similar UIs tend to request same permissions for similar purposes, such as camera for QR code scanning [9].

Reference Descriptions. Existing approaches [9], [8] typically rely on short sentences extracted from app descriptions as ground truth, which are often brief and lack behavior details. To ensure high quality, we constructed comprehensive reference descriptions by systematically annotating full app descriptions and privacy policies, supplemented by web searches and hands-on testing when needed. Our annotators were undergraduate students with substantial Android experience. Annotations were validated through a two-person review with arbitration by the project lead. The annotator tutorial and detailed results are available at our project website [30].

Constructing PP-KB. As shown in Table I, our PP-KB contains 3,500 app samples across seven permission types, with 500 apps per permission. We collected popular apps from Google Play focusing on apps that request at least one target permission. Unlike the evaluation dataset, we did not filter PP-KB apps based on privacy policy or description quality. The PP-KB uses LLM-generated descriptions that prioritize coverage and diversity over strict accuracy.

Metrics. Traditional syntactic evaluation metrics, such as BLEU [43] and ROUGE [44], are not adequate for our evaluations since they mainly measure lexical similarity. Given that permission descriptions may be semantically equivalent despite using different wordings, these metrics do not fully align with our evaluation objectives [45]. To more effectively assess the quality of APPBDS's outputs in comparison to

reference descriptions, we define two types of evaluation metrics: (1) semantic metrics, and (2) quality metrics.

- *Semantic Metrics:* We employ two semantic metrics. First, we use *BERT-Score*[46], which measures contextual embedding similarity. Since BERT-Score treats all textual content equally and does not prioritize permission-specific information, we additionally employ LLMs for automated evaluation of the generated descriptions. Recent studies have shown LLMs to be effective for NLG evaluation [27], [28], [29], allowing scalable evaluation without manual intervention. To this end, we propose *Semantic Relatedness*, an LLM-based metric that extracts permission-related semantic key points from the reference descriptions and measures how well the generated descriptions capture these key points. The metric emphasizes the most functionally relevant aspects of permission use. It employs a 0–10 scoring scale, where LLMs first extract distinct functional aspects from the reference, then evaluate the degree of coverage (none/partial/full) for each aspect. Scores are weighted by the complexity and significance of each aspect, considering both semantic equivalence and implementation details.
- *Quality Metrics:* We further employ two LLM-based metrics (*Richness* and *Specificity*) to assess the information quality of the generated descriptions beyond simple reference matching. While using reference descriptions as baseline (scored as 7 out of 10), both metrics allow for content variety beyond the reference. *Richness* evaluates the breadth of information provided in the description, focusing on the variety of aspects covered regardless of their strict alignment with the reference. It measures how well the description captures different dimensions of permission usage, such as various features, scenarios, and implementation purposes. The metric uses a 0–10 scale where 0 indicates lack of functional diversity, 10 indicates comprehensive coverage of permission-related functionalities, and reference descriptions serve as the 7-point baseline. *Specificity* examines the detail level of the provided information. It assesses whether the description provides concrete implementation details, precise feature explanations, and clear technical mechanisms, rather than just high-level summaries. The metric uses a 0–10 scale where 0 indicates lack of detailed information, 10 indicates highly specific explanations for each functionality, and reference descriptions serve as the 7-point baseline.

To ensure fair evaluation across all methods, we designed a rigorous LLM-based refinement prompt to standardize all generated descriptions before scoring. This prompt enforces the removal of promotional or irrelevant content, constrains responses to a concise 75–150 word range, and ensures inclusion of only permission-relevant, substantive functionality details. This pre-evaluation step minimizes length-related bias across LLM-generated outputs, directing the evaluation toward content quality rather than superficial verbosity.

Manual Inspection. To further ensure the LLM’s auto-evaluation provides accurate assessment, we conducted a manual inspection of the generated descriptions. We recruited three

TABLE II: Overall evaluation results

Method	BERT-Score	Richness	Specificity	Semantic Rel.
DescribeCTX	0.717	1.03	1.29	0.82
DescribeCTX _{GPT-4o}	0.832	6.33	6.22	3.99
<i>APPBDS with different LLMs</i>				
GPT-4o	0.875	8.54	7.96	6.20
GPT-oss 120B	0.764	8.52	9.02	6.17
LLaMA-3.3 70B	0.801	8.43	7.31	6.32
LLaMA-4 Maverick	0.810	7.86	7.49	5.85
Qwen-3 32B	0.810	8.79	8.86	6.70
DeepSeek-v3	0.762	7.93	8.29	5.95

TABLE III: Privacy-relevant node filtering results

	UCG	General	Proposition-Spec.	Similar-Apps-Spec.
# Nodes	46,282	43	29	21

inspectors with backgrounds in mobile security and software engineering. These inspectors used the same evaluation metrics provided to the LLM as guiding criteria for manual scoring, i.e., assessing the permission descriptions based on semantic relatedness, and richness and specificity.

To assess alignment between human and LLM evaluation, we calculated Pearson (r), Spearman (ρ), and Kendall’s Tau (τ) correlation coefficients. These complementary measures evaluate the consistency between LLM automated evaluation and human inspection. All coefficients range from -1 to 1, where values closer to 1 indicate stronger positive correlation (higher consistency between human and LLM judgments) and values near 0 suggest no correlation. Correlation values above 0.7 are generally considered strong, indicating reliable alignment between automated and manual assessments.

Due to the non-trivial efforts in inspecting the outputs from APPBDS and the baseline approaches, within our affordable efforts, we randomly chose 50 samples from all seven permissions to perform manual inspection. Among these 50 samples, except for the SMS permission, which had only two samples, we chose at least five samples from each permission.

Baseline Approaches. We selected two baseline approaches to compare with APPBDS. The first baseline approach is DescribeCTX [9], which is the state-of-the-art technique that trains an encoder-decoder model to synthesize permission descriptions for the requested permissions of apps. The second baseline approach is DescribeCTX_{GPT-4o}, which directly prompts the GPT-4o model with the inputs used by DescribeCTX. We compared with this baseline approach to assess the pre-trained LLM’s inherent knowledge and capabilities in synthesizing permission descriptions for the declared permissions of apps.

Implementation Details. In our evaluations, the default backend of APPBDS is GPT-4o [17]. We also report results with five open-source backends: GPT-oss 120B [47], LLaMA-3.3 70B [48], [49], LLaMA-4 Maverick [50], Qwen-3 32B [51], and DeepSeek-v3 [52]. For PP-KB construction, we use llama-3.1-sonar-small-128k-online [53], [54] to search and collect app information for cost effectiveness, and GPT-4o for proposition generations, with text-embedding-3-small [55] as the 1536-d embedding model.

In Similar App Retrieval, we set the number of retrieved similar apps (k) to 4. In Privacy-Relevant Node Filter, re-

TABLE IV: Human inspection results and correlation with LLM evaluations

Metrics	Inspector 1			Inspector 2			Inspector 3			LLM Evaluation		
				DescribeCTX / DescribeCTX _{GPT-4o} / APPBDS								
Richness	2.06	5.74	7.96	2.16	7.40	8.22	3.12	6.62	8.16	1.00	6.49	8.44
Specificity	1.92	5.36	7.98	2.06	7.42	8.36	3.60	7.18	8.60	1.36	6.31	7.92
Semantic Rel.	2.06	4.80	6.68	2.04	7.30	8.28	4.22	7.82	8.70	1.02	4.35	6.04
Correlations with LLM Evaluation Results (Pearson r / Spearman ρ / Kendall τ)												
Richness	0.886 / 0.867 / 0.751			0.922 / 0.829 / 0.717			0.913 / 0.875 / 0.770			— / — / —		
Specificity	0.871 / 0.854 / 0.721			0.890 / 0.782 / 0.649			0.887 / 0.807 / 0.688			— / — / —		
Semantic Rel.	0.759 / 0.775 / 0.628			0.799 / 0.798 / 0.660			0.770 / 0.762 / 0.638			— / — / —		

garding Android Protected API calls, we manually collected 49 unique permission-protected APIs related to our seven focused permissions from the Android Developer website [56], [57] for API levels 21-34. Similarly, we collect 138 third-party SDKs from Google Play SDK Index [35], a repository that provides detailed information about SDKs commonly used in Android applications. For evaluation, we employ the `deberta-xlarge-mnli` model [58] to calculate BERT-Score and utilize GPT-4o [17] for all other LLM-based metrics.

B. RQ1: Effectiveness and Improvements

We compare APPBDS with two baselines: (1) `DescribeCTX` [9], a state-of-the-art encoder-decoder approach for synthesizing descriptions of app permission uses, and (2) `DescribeCTXGPT-4o`, an enhanced version that replaces the encoder-decoder model with an LLM. As shown in Table II, APPBDS achieves the best results across all evaluation metrics. Compared to `DescribeCTX`, it improves BERT-Score by 22.0% (0.875 vs. 0.717), and achieves even larger gains in the LLM-based metrics. Compared to `DescribeCTXGPT-4o`, APPBDS shows consistent improvements, including a 5.2% improvement in BERT-Score (0.875 vs. 0.832), a 34.9% improvement in Richness (8.54 vs. 6.33), a 28.0% improvement in Specificity (7.96 vs. 6.22), and a 55.4% improvement in Semantic Relatedness (6.20 vs. 3.99). These results demonstrate that combining program analysis with LLMs produces significantly better descriptions than either encoder-decoder training or a straight-forward LLM baseline. The manual inspection results in Table IV further support our findings. All three human inspectors rated APPBDS significantly higher than both baselines across all metrics, with particularly notable superiority in Semantic Relatedness.

We further evaluate APPBDS’s effectiveness across a range of open-source LLMs. The results show that all open source LLMs achieve comparable results as GPT-4o, with some open source models leading on specific metrics, such as `Qwen-3 32B` for Richness and Semantic Relatedness. These outcomes indicate that APPBDS’s gains stem from the pipeline rather than any single model, underscoring its model-agnostic design. We conducted human inspection on a stratified subset covering all seven permissions and all five open-source LLM backends. Inspectors judged the generated descriptions to be of consistently high quality and aligned with the LLM-based evaluation across all metrics.

Effectiveness of Privacy-Relevant Node Filter. Table III compares the number of nodes in the original UCG with

TABLE V: Results of APPBDS and its ablation variants

Method	BERT-Score	Richness	Specificity	Semantic Rel.
APPBDS	0.875	8.54	7.96	6.20
APPBDS _{NoPatterns}	0.856	7.70	6.80	4.28
APPBDS _{NoGen&Prop}	0.809	7.38	7.31	4.92
APPBDS _{NoGen&Sim}	0.834	7.21	7.04	4.63
APPBDS _{NoSim&Prop}	0.839	8.60	8.20	6.14

those in the subgraphs generated after Privacy-Relevant Node Filter. The initial UCGs contain an average of 46,282 nodes, a scale beyond the processing capacity of even state-of-the-art LLMs. After filtering, the general subgraph is reduced to 43 nodes, while proposition-specialized and similar-apps-specialized subgraphs adding an average of 29 and 21 nodes, respectively. This step reduces information size by approximately 99.9% while preserving privacy-relevant content, greatly facilitating subsequent analysis.

Effectiveness of PP-KB in Supporting APPBDS. We conducted a targeted analysis comparing propositions generated from two sources: (1) the app’s PP and (2) the information retrieved from similar apps via the PP-KB. Our analysis of the complete evaluation dataset (270 apps) revealed that, for 28 apps, information from similar apps provided coverage of functionality that substantially exceeded the apps’ own propositions, revealing critical functionality entirely missing from the app’s own propositions. Moreover, in over 100 apps, information from similar apps meaningfully supplemented the original PP, adding context and specificity that improved understanding of permission usage scenarios.

Case Study. Figure 4 shows the reference description, APPBDS’s outputs, and baselines’ outputs for a team communication app, `co.bonx.go`, using the `MICROPHONE` permission. As shown in the figure, APPBDS effectively identifies two key functions described in the reference: voice activity detection for hands-free operation (I) and real-time group conversations (II). Additionally, APPBDS identifies a function not mentioned in the reference: “audio recording during video capture” (VII). By integrating these key points, APPBDS accurately summarizes the app’s core functionality: group talk and live casting, which we confirmed by checking the app’s official website [59]. For technical implementation details (marked in red in the reference), APPBDS partially addresses “testing audio input levels and managing earphone functionality” (V), but lacks depth regarding machine learning for sound environment adaptation and dual-microphone noise canceling technology (VI). This limitation occurs because evidence of these uncommon features is sparse across all information sources.

AppID: co.bonx.go — Permission: MICROPHONE
Reference
“...Voice Activity Detection I , which utilizes the microphone to detect when the user begins speaking, enabling hands-free operation by transmitting voice data only during active speech I . This feature employs machine learning to adapt to the sound environment V , ensuring precise voice detection with minimal processing time. ... The Push-to-Talk functionality transforms the app into a walkie-talkie, facilitating real-time group conversations II by efficiently capturing and transmitting voice data. ... The app implements dual-microphone digital noise-cancelling technology VI to filter out background noise and enhance sound clarity...”
DescribeCTX
“external microphone support”
DescribeCTX_{GPT}
“The app utilizes MICROPHONE data to record audio for the purpose of generating and displaying processed data as a spectrogram III or analyzer view. ... The user interface supports this functionality with options to start and stop recording IV , ensuring user control over when audio data is captured.”
APPBDS
“The app’s MICROPHONE permissions are integral to its core functionalities, enabling voice recognition and multimedia recording. Users can execute voice commands and perform searches hands-free I ... The app uses the microphone for testing audio input levels and managing earphone functionality V . Additionally, the app supports audio recording during video capture VII , allowing users to produce high-quality multimedia content with synchronized audio. For real-time communication, the app utilizes the microphone to facilitate live discussions within talk rooms II . ”

Fig. 4: Case study

Examining the baselines’ outputs reveals their limitations. With information from the privacy policy (mentioning “spectrogram or analyzer view” and “record audio for analysis purposes”) and UI elements (“start recording stop return”), DescribeCTX produces only a meaningless phrase (“external microphone support”), failing to identify the core functions. Meanwhile, DescribeCTX_{GPT-4o} focuses only on spectrogram visualization (**III**) and recording controls (**IV**) without discovering the app’s core permission-related functions. These results illustrate a common issue with baselines: when privacy policies are of poor quality or UI information is inadequate, they either make unfounded speculations or miss core functions. In contrast, APPBDS’s robust information retrieval capabilities remain effective even with limited initial information, demonstrating its significant advantages over the baselines.

C. RQ2: Human-LLM Evaluation Alignment

RQ2 assess the reliability of our LLM-based evaluation metrics. This analysis is critical as recent work has shown varying degrees of consistency between LLM evaluations and human assessments [27], [28]. We calculated correlation coefficients across the selected samples between human inspectors and LLM metrics. For each inspector, we paired their ratings with corresponding LLM scores for each metric (Richness, Specificity, and Semantic Relatedness) across all approaches. We computed three correlation coefficients: (1) Pearson correlation coefficient (r) for measuring linear relationship strength, (2) Spearman’s rank correlation (ρ) for measuring monotonic relationship strength, and (3) Kendall’s Tau (τ) for measuring concordance between rankings.

Table IV presents both the inspection scores and the correlation analysis results. The results demonstrate a strong alignment between human and LLM judgments in all metrics. First, inspectors’ relative rankings consistently match LLM evaluations, with APPBDS receiving the highest scores, followed by DescribeCTX_{GPT-4o} and DescribeCTX. The correlation anal-

ysis shows that Pearson correlation coefficients range from 0.759 to 0.922, indicating strong positive correlations (values above 0.7). Similarly, Spearman’s rank correlations range from 0.762 to 0.875, while Kendall’s Tau values fall between 0.628 and 0.770, both exceeding the 0.6 threshold for strong rank-order correlations. The p-values for all coefficients are $\ll 0.001$, indicating strong statistical significance. The consistently high correlations across different inspectors confirm that our LLM-based metrics effectively serve as reliable proxies for human judgment in assessing permission descriptions.

D. RQ3: Ablation Study

To evaluate the effectiveness of each component of our pipeline, we designed several ablation settings by creating variants of APPBDS, as shown in Table V:

- APPBDS_{NoPatterns}: Replaces privacy-relevant node filter with random node selection in Phase II, and replaces all subgraphs in Phase III with random subgraphs.
- APPBDS_{NoGen&Prop}: Removes both Agent 1 Permission-Focused Analysis and Agent 2 Policy Proposition Validation in Phase III, utilizing only the similar apps’ information, Agent 3 Similar-App-Inspired Discovery and Aggregator in Phase III.
- APPBDS_{NoGen&Sim}: Removes both Agent 1 and Agent 3 in Phase III, utilizing only permission-related propositions, Agent 2 and Aggregator in Phase III.
- APPBDS_{NoSim&Prop}: Removes permission proposition inference and similar app retrieval, discarding their specialized subgraphs and utilizing only the Agent 1 and Aggregator in Phase III.

Our ablation study reveals key insights about APPBDS’s component contributions. As shown in Table V, APPBDS_{NoPatterns} achieves the lowest scores in specificity and semantic relatedness among all variants, with semantic relatedness only slightly surpassing the DescribeCTX_{GPT-4o} baseline (4.28 vs 3.99). This occurs due to the sparse

distribution of relevant information within app code, making random node selection likely to capture irrelevant or insufficient permission-related details. APPBDS_{NoGen&Prop} focuses exclusively on discovering features mentioned in similar apps, while APPBDS_{NoGen&Sim} prioritizes verifying privacy policy claims. This narrow focus results in descriptions lacking general and common functionality, reflected in their lower semantic relatedness scores. While APPBDS attains the highest BERT-score and semantic relatedness score across all ablation settings, APPBDS_{NoSim&Prop} exhibits slightly higher richness and specificity scores, with a semantic relatedness score comparable to APPBDS. APPBDS_{NoSim&Prop} relies solely on the general subgraph, which primarily identifies directly observable permission-related functionality. Though it generates rich and specific descriptions due to detailed subgraph node evidence, it misses subtle or implicit permission uses that are not immediately apparent from code patterns alone.

To illustrate these differences, consider a taxi app using the LOCATION permission. In this scenario, APPBDS_{NoSim&Prop} identifies direct location-related functions such as real-time location tracking and address search capabilities, along with evidence of Google location services integration. However, it fails to capture the functional context of how these capabilities integrate with the app's purpose and user interactions. In contrast, APPBDS_{NoGen&Prop} and APPBDS_{NoGen&Sim} complement these findings by accurately identifying driver-passenger matching based on location, trip tracking visualization, fare calculation, location-based service personalization, and dynamic UI adjustments using location data. These insights enrich the generated descriptions with valuable contextual understanding.

These findings demonstrate that while general subgraph analysis provides a strong foundation, the complete APPBDS pipeline's integration of privacy policy propositions and similar apps' information along with specialized subgraphs yields the most semantically accurate and comprehensive permission descriptions. The next section provides a detailed case study, demonstrating how APPBDS effectively identifies core permission-related functionalities and outperforms baseline approaches in a real-world scenario.

E. RQ4: Robustness Against Code Obfuscation

To evaluate APPBDS's robustness against code obfuscation, we conducted two more empirical studies. First, we classified apps in our 270-app test dataset based on their existing obfuscation status by manually examining the decompiled code signatures and method bodies to classify apps into two categories: 86 natively obfuscated (N-Obf) apps where all method names are obfuscated, and 184 natively clear (N-Clear) apps that retain readable method names. Figure 3 shows the classification results, indicating that popular apps in larger download count buckets are more likely to employ method-name obfuscation. We then compares APPBDS's performance on N-Obf apps and N-Clear apps, as shown in Table VI. The results confirm that APPBDS is largely unaffected by method-name obfuscation: performance on N-Obf apps is only slightly

TABLE VI: Performance comparison between natively obfuscated apps (N-Obf, n=86) and natively clear apps (N-Clear, n=184).

Backend Model	BERT-Score		Richness		Specificity		Semantic Rel.	
	N-Obf	N-Clear	N-Obf	N-Clear	N-Obf	N-Clear	N-Obf	N-Clear
GPT-4o	0.879	0.873	8.45	8.58	7.84	7.98	6.08	6.25
GPT-oss 120B	0.773	0.760	8.43	8.56	8.93	9.06	6.48	6.00
LLaMA-3.3 70B	0.805	0.810	8.30	8.49	7.13	7.39	6.21	6.37
LLaMA-4 Maverick	0.794	0.818	7.79	7.88	7.32	7.51	5.80	5.85
Qwen-3 32B	0.803	0.813	8.74	8.78	8.82	8.86	6.66	6.74
DeepSeek-v3	0.773	0.758	7.78	8.00	8.34	8.27	5.91	5.97
Average	0.804	0.805	8.25	8.38	8.06	8.18	6.19	6.20

TABLE VII: Performance before and after applying Obfuscapk tool (Raw vs E-Obf, n=131 successfully processed apps).

Backend Model	BERT-Score		Richness		Specificity		Semantic Rel.	
	E-Obf	Raw	E-Obf	Raw	E-Obf	Raw	E-Obf	Raw
GPT-4o	0.853	0.868	8.37	8.61	7.44	7.99	4.79	6.25
GPT-oss 120B	0.768	0.763	8.23	8.44	9.02	9.01	4.63	6.07
LLaMA-3.3 70B	0.797	0.797	8.32	8.39	6.79	7.32	5.31	6.23
LLaMA-4 Maverick	0.791	0.799	7.56	7.93	7.05	7.49	4.75	5.85
Qwen-3 32B	0.800	0.815	8.72	8.81	8.75	8.82	5.60	6.79
DeepSeek-v3	0.762	0.758	7.60	7.95	7.89	8.35	4.60	6.03
Average	0.795	0.800	8.13	8.36	7.82	8.16	4.95	6.20

lower than on N-Clear apps, with consistent patterns across all backends.

Second, we applied Obfuscapk [60], a widely used open-source framework in Android security research, on the test apps to assess the impact of aggressive obfuscation. Obfuscapk decompiled each APK, applied class/method/field-renaming, and then rebuilt, aligned, and re-signed the package. We successfully obfuscated 131 out of 270 apps, enabling a controlled comparison between the raw versions (Raw) and their obfuscated counterparts (E-Obf). As shown in Table VII, APPBDS maintains robust performance on E-Obf apps for the Richness and the Specificity scores (averagely <0.34 drops), and exhibits slightly greater degradation for the Semantic Relatedness score (about 20% drops). The findings suggest that, since most of APPBDS's information sources (e.g., framework APIs, string resources, and PP) cannot be obfuscated, they remain highly resilient to aggressive transformations; the main impact of obfuscation lies in degrading the semantic coherence of the generated descriptions. We further analyze the impact of less common obfuscation schemes on APPBDS's components in Section VI.

F. RQ5: Error Taxonomy and Analysis

We inspected the incorrect generated descriptions and identified four major types of errors produced by APPBDS: incompleteness (Omission), imprecision (Generalization), minor inaccuracies (Factual Detail Error), and direct factual opposition (Contradiction). Our analysis of 1,322 discrepancies in the generated descriptions revealed that the predominant failure patterns are Omission (790) and Generalization (496). Notably, <3% of errors are more severe errors (15 for Factual Detail Error and 21 for Contradiction), indicating the high factual reliability of APPBDS. We then performed a root cause analysis on these errors and identified two major causes: evidence-related (insufficient/ambiguous input) and synthesis-related (overly cautious generalization). Results show that most errors were caused by evidence-related issues, including omissions,

factual inaccuracies, contradictions, and a significant fraction of generalization failures (~40%). Overly cautious generalization emerged as the second most common cause, often reflecting the model’s attempt to navigate against unclear evidence. Failures from unprompted model hallucination were rare, confirming that our agent design effectively anchors the model to available evidence. Thus, the primary challenge is the completeness of the evidence collection pipeline, as the synthesis component itself has proven highly robust against generating ungrounded factual errors.

V. THREATS TO VALIDITY

Internal Validity. Our study faces potential internal threats from three sources. First, subjective annotation biases could affect reference descriptions. To mitigate this, we implemented a dual-annotator validation process where initial descriptions were cross-checked against multiple evidence sources, including app functionality verification and official documentation. Second, LLM generation randomness could lead to inconsistent outputs. We addressed this by applying systematic normalization throughout the description generation process to eliminate speculative content and maintain consistent output quality. Third, manual inspection reliability might vary across inspectors. We mitigated this by using standardized evaluation prompts with explicitly defined dimensions and identical metric prompts to ensure alignment between automated and manual assessments. Additionally, we conducted inspections following stratified sampling principles to ensure balanced representation across permission types.

External Validity. The generalizability of our findings is mainly limited by two factors. First, our selection of permissions might not cover all possible use cases. We addressed this by carefully selecting permissions based on their critical relevance to user privacy and functional necessity, focusing on widely recognized high-risk categories. Second, our evaluation apps might not represent the entire app ecosystem. To mitigate this, we incorporated apps spanning multiple categories from mainstream sources to reflect real-world app diversity, prioritizing popular apps while maintaining broad functional coverage. Our methodology is designed to be adaptable to broader contexts, including additional permission types and underrepresented app categories.

VI. DISCUSSION

Code Obfuscation. APPBDS is most vulnerable when obfuscation removes or distorts static evidence required to build and validate the UCG. High-risk schemes include (i) reflection and dynamic loading that hide permission-protected API calls; (ii) string/resource encryption that suppresses UI and constant-level cues; (iii) third-party SDK renaming that breaks SDK identification; and (iv) heavy control-flow transformations that degrade decompilation and graph construction. Mitigations include integrating de-obfuscation and reflection/JNI resolution before Phase I [61], [62], [63], introducing lightweight runtime sampling to recover decrypted strings/UI texts, canonicalizing SDK fingerprints beyond package names, and enforcing stricter evidence gating in the Aggregator.

PP-KB and Retrieval Mechanism. APPBDS’s similarity measurement relies on proposition-level semantic similarity, which may not capture functional similarity between apps with similar policy language but different permission implementations. Also, the current PP-KB (3,500 apps) may not cover emerging app categories. In future work, we may employ advanced similarity metrics that combine app metadata and proposition embeddings, along with continuous PP-KB expansion.

VII. RELATED WORK

App Permission Description Generation. Several approaches extract explanatory sentences from app descriptions to explain permission uses [8], [11], though these typically fail to cover all declared permissions. Other work [14] employs behavior description models for sensitive API call graphs, relying solely on code without considering privacy policies or GUI information. Similarly, some research [64] identifies sentences describing types of information used in apps. Most closely related to our work, [9] generates permission descriptions using similar input resources but employs an encoder-decoder model limited by short token sequences. Our evaluation demonstrates the superiority of LLMs over this approach.

Privacy Policy Analysis. Recent research leverages LLMs for automated privacy policy analysis, with approaches for identifying and classifying data practice disclosures [65] and privacy policy text analysis[66]. APPBDS analyzes not only privacy policies but also code information to generate functionality-based descriptions for permission uses.

Permission Usage Analysis. Recent research such as [67] proposes dynamic analysis tools for tracking permission usage behaviors and their contexts at runtime. Our work differs by using static analysis tools to build a comprehensive permission usage analysis model that combines code, UI, and policy information.

Privacy Compliance Verification. Recent work examines discrepancies between app behaviors and privacy disclosures, with approaches like [68] measuring non-compliance of Apple Privacy Labels and [69] detecting inconsistencies between privacy policies and labels. While these approaches focus on verification and compliance checking, APPBDS aims to generate natural language descriptions of permission uses that can help improve transparency.

VIII. CONCLUSION

In this paper, we presented APPBDS, a novel approach that integrates program analysis and LLMs to identify and summarize privacy-related code elements and UI contexts, generating detailed descriptions for apps’ permission uses. In particular, APPBDS constructs UCGs that integrate UI contexts with code semantics, employs critical node patterns to identify permission-related nodes, and leverages LLMs to extract essential privacy propositions from privacy policies. Additionally, APPBDS builds a PP-KB and identifies similar apps to guide LLMs when faced with uninformative privacy policies. We conducted extensive experiments on 270 real-world apps to validate the effectiveness of our approach and demonstrate its superiority over state-of-the-art techniques.

ACKNOWLEDGEMENT

Xusheng Xiao's work is partially supported by the National Science Foundation under the grants CCF-2318483.

REFERENCES

- [1] B. Liu, H. Jin, and R. Govindan, “Medusa: A programming framework for crowd-sensing applications,” in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2015, pp. 337–350.
- [2] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, “IconIntent: Automatic identification of sensitive ui widgets based on icon classification for android apps,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2019.
- [3] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, , and J. Lu, “DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.
- [4] Y. Zhou and X. Jiang, “Dissecting android malware: Characterization and evolution,” in *Proceedings of the IEEE Symposium on Security and Privacy (S & P)*, 2012, pp. 95–109.
- [5] P. G. Kelley, J. Bresee, L. F. Cranor, and R. W. Reeder, “A “nutrition label” for privacy,” in *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2009.
- [6] P. G. Kelley, L. Cesca, J. Bresee, and L. F. Cranor, “Standardizing privacy notices: an online study of the nutrition label approach,” in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2010.
- [7] Apple Inc., “Human Interface Guidelines: Requesting Permission,” [Online]. Available: <https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/requesting-permission/>, accessed: Feb. 7, 2020.
- [8] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, “WHYPER: towards automating risk assessment of mobile applications,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2013.
- [9] S. Yang, Y. Wang, Y. Yao, H. Wang, Y. F. Ye, and X. Xiao, “Describectx: context-aware description synthesis for sensitive behaviors in mobile apps,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2022.
- [10] X. Liu, Y. Leng, W. Yang, W. Wang, C. Zhai, and T. Xie, “A large-scale empirical study on android runtime-permission rationale messages,” in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018, pp. 137–146.
- [11] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, “Autocog: Measuring the description-to-permission fidelity in android applications,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.
- [12] M. L. V. de Vanter, “The Documentary Structure of Source Code,” *Information & Software Technology*, vol. 44, no. 13, 2002.
- [13] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, “Towards automatically generating summary comments for java methods,” in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2010.
- [14] M. Zhang, Y. Duan, Q. Feng, and H. Yin, “Towards automatic generation of security-centric descriptions for android apps,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.
- [15] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, “Neural-machine-translation-based commit message generation: how far are we?” in *Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 2018.
- [16] A. LeClair, S. Jiang, and C. McMillan, “A neural model for generating natural language summaries of program subroutines,” in *Proceedings of the ACM/IEEE International Conference on Software Engineering (ICSE)*, 2019, pp. 795–806.
- [17] OpenAI, “GPT-4o,” [Online]. Available: <https://openai.com/index-hello-gpt-4o/>, 2024, accessed: 2024.
- [18] OpenAI, J. Achiam, S. Adler *et al.*, “GPT-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2024.
- [19] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal *et al.*, “Language models are few-shot learners,” *arXiv preprint arXiv:2005.14165*, 2020.
- [20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
- [21] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal *et al.*, “Retrieval-augmented generation for knowledge-intensive nlp tasks,” *arXiv preprint arXiv:2005.11401*, 2021.
- [22] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, “Policylint: Investigating internal privacy policy contradictions on google play,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2019.
- [23] L. Li, A. Bartel, T. F. D. A. Bissyande, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, “IccTa: detecting inter-component privacy leaks in android apps,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015.
- [24] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, “Appcontext: Differentiating malicious and benign mobile app behavior under contexts,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015.
- [25] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, “Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2014.
- [26] Y. Hu, H. Wang, T. Ji, X. Xiao, X. Luo, P. Gao, and Y. Guo, “Champ: Characterizing undesired app behaviors from user comments based on market policies,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2021.
- [27] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, “G-eval: NLG evaluation using gpt-4 with better human alignment,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- [28] J. Wang, Y. Liang, F. Meng, Z. Sun, H. Shi, Z. Li, J. Xu, J. Qu, and J. Zhou, “Is ChatGPT a good NLG evaluator? a preliminary study,” in *Proceedings of the New Frontiers in Summarization Workshop at the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- [29] Y.-T. Lin and Y.-N. Chen, “LLM-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models,” in *Proceedings of the NLP for Conversational AI Workshop (NLP4ConvAI) at the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.
- [30] “APPBDS project website,” <https://github.com/AppBDS/AppBDS>, 2025.
- [31] R. Vallee-Rai, E. Gagnon, L. Hendren, P. Lam, P. Pominville, and V. Sundaresan, “Optimizing java bytecode using the soot framework: Is it feasible?” in *Proceedings of the International Conference on Compiler Construction (CC)*, 2000.
- [32] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, “Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps,” in *Proceedings of the ACM International Conference on Programming Language and Design Implementation (PLDI)*, 2014.
- [33] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon, “Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis,” in *Proceedings of the USENIX Conference on Security (USENIX Security)*, 2013.
- [34] Google, “Google Play Store,” [Online]. Available: <https://play.google.com/store?hl=en>, 2024, accessed: 2024.
- [35] Google, “Google Play SDK Index,” [Online]. Available: <https://play.google.com/sdk/>, 2025, accessed: 2025.
- [36] Z. Chen, Z. Jiang, F. Yang, E. Cho, X. Fan, X. Huang, Y. Lu, and A. Galstyan, “Graph meets LLM: A novel approach to collaborative filtering for robust conversational understanding,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Industry Track (EMNLP)*, 2023.
- [37] J. Huang, X. Zhang, Q. Mei, and J. Ma, “Can llms effectively leverage graph structural information: When and why,” *arXiv preprint arXiv:2309.16595*, 2023.
- [38] B. Andow, A. Acharya, D. Li, W. Enck, K. Singh, and T. Xie, “UiRef: Analysis of sensitive user inputs in android applications,” in *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2017.
- [39] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, “WHYPER: Towards

- automating risk assessment of mobile applications,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2013.
- [40] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov, “The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences,” in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2017.
- [41] P. Wijesekera, J. Reardon, I. Reyes, L. Tsai, J.-W. Chen, N. Good, D. Wagner, K. Beznosov, and S. Egelman, “Contextualizing privacy decisions for better prediction (and protection),” in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2018.
- [42] K. Alix, T. F. Bissyandé, J. Klein, and Y. Le Traon, “Androzoo: Collecting millions of android apps for the research community,” in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2016.
- [43] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.
- [44] C.-Y. Lin, “ROUGE: A package for automatic evaluation of summaries,” in *Proceedings of the Text Summarization Branches Out Workshop at the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2004.
- [45] A. B. Sai, A. K. Mohankumar, and M. M. Khapra, “A survey of evaluation metrics used for nlg systems,” *ACM Computing Surveys*, vol. 55, no. 2, pp. 26:1–26:39, 2022.
- [46] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, “Bertscore: Evaluating text generation with bert,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.
- [47] OpenAI, “gpt-oss-120b & gpt-oss-20b model card,” *arXiv preprint arXiv:2508.10925*, 2025.
- [48] Meta AI, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
- [49] “Llama-3.3-70b-instruct,” Hugging Face, Meta AI, 2024, accessed: 2025-09-12. [Online]. Available: <https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct>
- [50] “Llama-4-Maverick-17B-128E,” Hugging Face, Meta AI, 2025. [Online]. Available: <https://huggingface.co/meta-llama/Llama-4-Maverick-17B-128E>
- [51] Qwen Team, “Qwen3 technical report,” *arXiv preprint arXiv:2505.09388*, 2025.
- [52] DeepSeek-AI, “Deepseek-v3 technical report,” *arXiv preprint arXiv:2412.19437*, 2024.
- [53] Meta, “Llama 3.1: Open Foundation and Fine-Tuned Chat Models,” [Online]. Available: <https://ai.meta.com/llama/>, 2024, accessed: 2024.
- [54] Perplexity AI, “Llama 3.1 Sonar Small 128K Online,” [Online]. Available: <https://www.perplexity.ai/>, <https://www.promptitude.io/models/llama-3-1-sonar-small-online>, 2024, accessed: 2024.
- [55] OpenAI, “text-embedding-3-small,” [Online]. Available: <https://openai.com/index/new-embedding-models-and-api-updates/>, 2024, accessed: 2024.
- [56] Google, “Android Developers,” [Online]. Available: <https://developer.android.com/>, 2024, accessed: 2024.
- [57] Google, “Android Permission Groups,” [Online]. Available: https://developer.android.com/reference/android/Manifest.permission_group, 2024, accessed: 2024.
- [58] P. He, X. Liu, J. Gao, and W. Chen, “Deberta: Decoding-enhanced bert with disentangled attention,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [59] BONX Inc., “Features - bonx work,” 2024, accessed: 2024-03-08. [Online]. Available: <https://bonx.co/work/en/service/features/>
- [60] S. Aonzo, G. C. Georgiu, L. Verderame, and A. Merlo, “Obfuscator: An open-source black-box obfuscation tool for android apps,” *SoftwareX*, vol. 11, p. 100403, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352711019302791>
- [61] B. Bichsel, V. Raychev, P. Tsankov, and M. Vechev, “Statistical deobfuscation of android applications,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.
- [62] G. You, G. Kim, S. Han, M. Park, and S.-J. Cho, “Deoptfuscator: Defeating advanced control-flow obfuscation using android runtime (ART),” *IEEE Access*, vol. 10, pp. 61 426–61 440, 2022.
- [63] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. Wang, and K. Zhang, “Understanding android obfuscation techniques: A large-scale investigation in the wild,” *arXiv preprint arXiv:1801.01633*, 2018.
- [64] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman, “Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with policheck,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2020.
- [65] D. Rodriguez, I. Yang, J. M. Del Alamo, and N. Sadeh, “Large language models: A new approach for privacy policy analysis at scale,” *arXiv preprint arXiv:2405.20900*, 2024.
- [66] C. Tang, Z. Liu, C. Ma, Z. Wu, Y. Li, W. Liu, D. Zhu, Q. Li, X. Li, T. Liu, and L. Fan, “Policygpt: Automated analysis of privacy policies with large language models,” *arXiv preprint arXiv:2309.10238*, 2023.
- [67] M. Guerra, R. Milanese, R. Oliveto, and F. Fasano, “Rpedroid: Runtime identification of permission usage contexts in android applications,” in *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2023.
- [68] Y. Xiao, Z. Li, Y. Qin, X. Bai, J. Guan, X. Liao, and L. Xing, “Lalaine: Measuring and characterizing non-compliance of apple privacy labels,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2023.
- [69] A. Jain, D. Rodriguez, J. M. Del Alamo, and N. Sadeh, “Atlas: Automatically detecting discrepancies between privacy policies and privacy labels,” *arXiv preprint arXiv:2306.09247*, 2023.