AgentDroid: A Multi-Agent Tool for Detecting Fraudulent Android Applications

Ruwei Pan
panruwei@stu.cqu.edu.cn
Chongqing University
Chongqing, China

Hongyu Zhang*
hyzhang@cqu.edu.cn
Chongqing University
Chongqing, China

Zhonghao Jiang
zhonghaojiang@cqu.edu.cn
Chongqing University
Chongqing, China

Ran Hou
houran@stu.cqu.edu.cn
Chongqing University
Chongqing, China

***Abstract*—With the increasing prevalence of fraudulent Android applications such as fake and malicious applications, it is crucial to detect them with high accuracy and adaptability. We present AgentDroid, a novel tool for Android fraudulent application detection based on multi-modal analysis and multi-agent systems. AgentDroid overcomes the limitations of traditional detection methods such as the inability to handle multimodal data and high false alarm rates. It processes Android applications and extracts a series of multi-modal data for analysis. Multiple LLM-based agents with specialized roles analyze the relevant data and collaborate to detect complex fraud effectively. We curated a dataset containing various categories of fraudulent applications and legitimate applications and validated our tool on this dataset. Experimental results indicate that our multi-agent tool based on GPT-4o achieves an accuracy of 91.7% and an F1-Score of 91.68%, outperforming the baseline methods. A video of AgentDroid is available at https://youtu.be/YOM9Ex-nBts.**

***Index Terms*—Android Fraud Detection, Multi-Agent Systems, Multi-Modal Analysis**

I. Introdution

As the Android app market continues to grow, fraudulent apps have become an increasingly significant security threat. Various types of fraudulent applications—a subset of malicious applications involving scam, sex, and gambling—can compromise data integrity, disrupt system availability, and leak private data[6]. For instance, research shows that ad fraud alone costs mobile advertisers up to $1.3 billion in 2015[4], and this number has likely increased in recent years. Thus, it is critical to develop innovative and adaptive solutions to identify and detect fraudulent Android applications[1].

A variety of methods [3] have been developed for detecting fraudulent applications, including feature extraction and analysis mechanisms. These methods collect characteristics of Android applications through static analysis tools and analyze the extracted data to detect fraudulent applications without executing Android package files. Android Package (APK) files contain multimodal data, including MD5 hashes, network addresses, operating system permissions, icons, and more. Each of these data types can provide crucial insights into fraudulent behavior. For example, fraudulent applications often use generic or low-quality icons that do not align with their advertised functionality or purpose, which can serve as an indicator of fraud. However, there are various limitations in detecting fraudulent applications using traditional static analysis [1], such as high false positive rates, poor generalization of machine learning models, and the inability to process multimodal data.

To address these challenges, we develop AgentDroid, a tool for detecting fraudulent applications based on multimodal analysis and a multi-agent system. The multi-agent system leverages multimodal analysis to integrate information from multiple data sources (such as fingerprint information and application icons) and combines the collaborative work of multiple agents to achieve a comprehensive understanding of application behavior. Our tool, AgentDroid, assigns agents specialized in analyzing multimodal data. These agents work in parallel and share findings with a Task Master agent, which coordinates their actions and consolidates results. In our experiments, we validated AgentDroid using a dataset of over 600 APK files, including both fraudulent and legitimate applications. The experimental results show that our approach can reduce the false alarm rate and improve detection accuracy.

We summarize our contributions as follows:

- We propose **AgentDroid**, a multi-agent collaborative tool that leverages **multi-modal data and specialized LLM-driven agents**, enabling accurate detection of fraudulent Android applications. Unlike traditional single-model approaches, AgentDroid can effectively handle heterogeneous features such as text, icons, and certificates in a unified and interpretable framework.
- We evaluate the effectiveness of AgentDroid through experiments, and the results show that our tool outperforms existing methods.
- We have open-sourced our tool and dataset at https://github.com/Wwstarry/LLM4Fraud.

II. Related Work

Early approaches detect Android malware using static features and machine learning algorithms, including Drebin[1], which extracts permissions, API calls, and other manifest-based features to train a linear SVM classifier for fraudulent detection. Meanwhile, deep learning-based models have been widely explored. For instance, Yuan et al. [10] implement an online deep-learning-based Android malware detection engine (DroidDetector) that can automatically detect whether an app is fraudulent or not. Recent methods such as AppPoet [11] and LAMD [7] use LLMs to process multimodal app data through prompt-based reasoning. AppPoet statically extracts multi-view features of apps and uses prompt-guided LLMs with a DNN classifier to semantically analyze and detect fraudulent Android applications, and LAMD is a context-driven framework that enables effective Android fraudulent detection by guiding LLMs through tiered reasoning over security-critical code regions. Unlike existing models, our tool adopts a collaborative multi-agent design, enabling specialized agents to reason over different data types and improve interpretability and adaptability.

III. Proposed Approach

We propose AgentDroid, a multi-agent fraud detection tool that uses static analysis and multimodal features extracted from APK files. Figure 1 provides an overview of AgentDroid. The tool analyzes metadata, permissions, icons and source code to determine the presence of fraudulent behavior and classify the app into specific fraud categories such as gambling and so on.

A. Static Analysis and Agent Tools

AgentDroid performs static analysis on each APK using Androguard [2], extracting multimodal data such as metadata, permissions, icons, and decompiled source code. Specifically, we analyze the DEX bytecode, the AndroidManifest.xml, the resource files, and so on. These features are then processed by specialized agents to detect whether the app is fraudulent.

To support this analysis, AgentDroid provides agents with querying interfaces for interpreting metadata, image classifiers (e.g., DeiT model [9]) for visual features (icons), and text classifiers (e.g., T5 model [8]) for textual features (permissions, descriptions) to inspect for fraudulent patterns. Source code is made accessible via a structured interface, allowing agents to search, navigate, and interpret program logic. Each agent is equipped with domain-specific tools and collaborates within the multi-agent system to contribute findings for the final decision-making.

B. Phase I Task Allocation

After Static Analysis and Agent Tools Training, the agents collaborate according to a pre-designed task allocation strategy, described in Algorithm 1. Firstly, the Task Master receives initial APK features and assigns tasks to specialized analytical agents. Each agent is responsible for a specific task and can utilize specific tools to perform the required analysis. The list of agents in our approach, all powered by LLMs, is as follows: ❶ **Task Master**: Assigns tasks to the appropriate agents and integrates their results for decision-making. ❷ **Package Tracer**: Retrieves the app package information, including the app name, activities, and related metadata. ❸ **Icon Analyst**: Analyzes the app's icon to detect if it resembles icons from known fraudulent apps. ❹ **Permission Analyst**: Analyzes the app's requested permissions to assess potential risks or sensitive behaviors. ❺ **Content Analyst**: Analyzes textual content within the app for any suspicious or illicit activity indicators. ❻ **Certificate Checker**: Verifies the authenticity and validity of the app's certificate to identify possible fraud. ❼ **Link Analyst**: Inspects relationships between the given app and other related apps to uncover hidden connections. ❽ **Decision Maker**: Aggregates all results and provides the final decision on whether the app is fraudulent.

Algorithm 1 Dynamic Task Allocation by Task Master

```
Require: Initial data $collectedData$
Ensure: Final result $finalResult$
 1: Initialize $tasksToAssign \leftarrow$ Determine initial tasks based
    on $collectedData$
 2: Initialize $collectedResults \leftarrow$ empty set
 3: while cannot-make-decision($collectResults$) do
 4:     for each $task$ in $tasksToAssign$ do
 5:         $agent \leftarrow$ Select the corresponding agent($task$)
 6:         Assign $task$ to $agent$
 7:         $result \leftarrow$ Receive result from $agent$
 8:         Add $result$ to $collectedResults$
 9:     Integrate $collectedResults$ and update $collectedData$
10:     $tasksToAssign \leftarrow$ Analyze and determine new tasks
    based on the updated $collectedData$
11: Integrate all $collectedResults$
12: Pass the integrated information to the final agent
    $finalAgent$
13: $finalResult \leftarrow$ Receive final result from $finalAgent$
14: return $finalResult$
```

Although some agents, such as the Permission Analyst and Content Analyst, both operate on textual inputs, we design them as separate modules due to their distinct data modalities and domain-specific reasoning requirements. Permission data is structured and typically requires an understanding of Android API semantics, while content analysis involves unstructured, natural-language understanding of user-facing elements such as descriptions or messages. Keeping these roles separate allows each agent to employ optimized tools (e.g., fine-tuned T5 for structured text) and improves adaptability to evolving fraud patterns.

The eight agents are broadly divided into two categories: one responsible for decision-making and the other responsible for analysis. As illustrated in Figure 1, the decision-making agents include the Task Master and the Decision Maker, while the analytical agents include the Certificate Checker, the Package Tracker, the Link Analyst, the Permission Analyst, the Icon Analyst, and the Content Analyst. All the agents in AgentDroid are powered by the GPT-4o model by default, which allows them to perform advanced reasoning and collaboration.

C. Phase II Multimodal Analysis

In the Multimodal Analysis, which is the core phase of our tool consisting of specialized agents collaborating for fraud detection, we use LangGraph and Prompt Engineering to create eight specialized agents responsible for task scheduling, icon analysis, content analysis, certificate inspection, package tracking, sensitive permission analysis, relationship analysis, and final decision-making. An example of the prompt design used to guide the agents is shown in Figure 2, which illustrates the role-playing structure, task description, and allowed tools

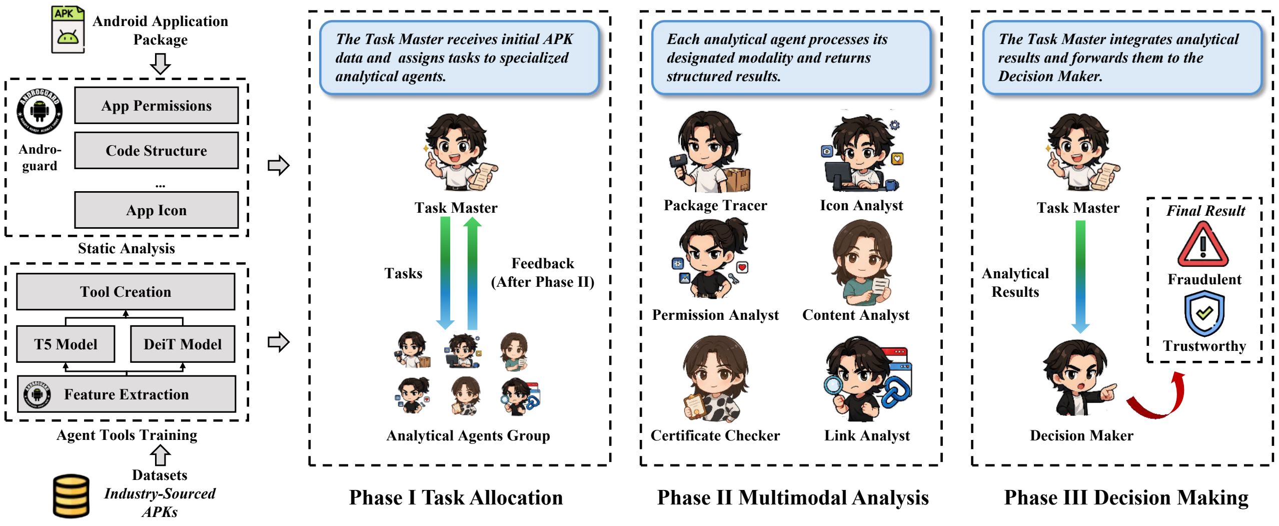

Fig. 1: An Overview of AgentDroid

for the Icon Analyst. Each analytical agent, such as the Package Tracker or Icon Analyst, specializes in processing specific types of data, such as certificates, permissions, icons, and content. Upon completing their tasks, the analytical agents return their results to the Task Master for integration. The Task Master orchestrates the process by determining whether additional information is needed based on collected data and assigns specific tasks to the analytical agents. The collaboration process among agents is coordinated by the Task Master, which maintains a shared knowledge state (`collectedData`) that aggregates intermediate results. When an agent completes a task, it returns structured output back to the Task Master. This output is appended to the shared state and can trigger new tasks that depend on the updated context. For instance, if the Package Tracer detects suspicious app naming patterns, this information may prompt further investigation from the Link Analyst. All agents communicate asynchronously via the LangGraph backend, and intermediate states are stored in memory to support iterative reasoning.

D. Phase III Decision Making

In this phase, the Task Master passes the consolidated information to the Decision Maker, who evaluates the aggregated data to determine the final category and probability of fraud for the APK. The Task Master coordinates agent execution based on task dependencies and data availability. For instance, if an app lacks certificate information, the Certificate Checker is skipped to prevent unnecessary analysis. Likewise, if the Permission Analyst detects suspicious access (e.g., to SMS or contacts), the Task Master can prioritize content inspection by the Content Analyst. This design enables adaptive task sequencing and avoids redundant or irrelevant processing. The eight agents are designed based on common fraud behaviors, such as fake certificates, icon impersonation, permission abuse, and scam-related content. Each agent targets one specific fraud type to ensure clearer analysis. The tool is modular and supports adding new agents to detect emerging fraud patterns.

Prompt Design	Prompt Content	Example (Icon Analyst)
Role-Playing	You are a <Role>. You are <Professional Expertise>.	You are a scam-involved app. You are good at classifying the app through icons. Be careful not to over-rely on the model's results, as they may be inaccurate.
Task Description	<Task Description>	Call the icon analysis model and summarize its results.
Tool Usage	You are allowed to call the following tools. <Agent Tool>: <Tool Description>	You are allowed to call the following tools. 1. get_icon_prob: Obtain the APP icon corresponding to the MD5 belongs to the following five types of probability ['black' 'gamble' 'scam' 'sex' 'white']. ...

Fig. 2: Prompt Template used by AgentDroid. The complete prompt template is available at our repository.

IV. Evaluation

A. Experimental Setup

We curated a dataset consisting of 660 APKs, encompassing 480 fraudulent applications and 180 legitimate applications, shown in Figure 3. This dataset is designed to train and validate the performance of our multi-agent system in detecting Android fraudulent applications. The APK files were from a major telecommunications enterprise(China Mobile). Fraudulent labels were officially assigned by the enterprise based on real-world network activity, making the dataset both representative and reliable for fraud detection research. The original data is available at: https://github.com/Wwstarry/LLM4Fraud Before conducting the experiments, we performed a static analysis on the APKs to extract key features including metadata, permissions, and icons. The extracted features were then cleaned and processed, after which the dataset was split into

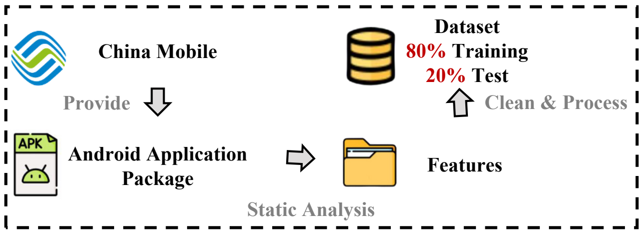

Fig. 3: The workflow of curating the dataset.

training (80%) and test (20%) sets to fine-tune and evaluate the performance of the tool across various modalities, including text and icon classification.

TABLE I: Comparison of Various Models for Fraudulent Android App Detection

Model	ACC(%)	Precision(%)	Recall(%)	F1(%)
BERT	44.53	46.19	44.53	45.34
Clip	33.33	31.50	33.33	32.39
Resnet-18 [5]	47.62	47.11	47.62	47.36
Drebin [1]	40.15	44.20	40.15	42.08
DroidDetector [10]	61.63	61.80	47.83	53.92
GPT-4o	78.09	79.91	81.02	80.46
AppPoet [11]	82.58	89.35	76.89	82.66
LAMD [7]	88.34	88.7	83.61	86.08
AgentDroid (GPT-4o)	**91.70**	**91.79**	**91.70**	**91.68**

In the experiments, all the LLMs in AgentDroid and baselines are powered by the GPT-4o model with a temperature of 0.5, which provides fast and accurate reasoning at a lower cost. We compare AgentDroid against several representative baselines, including machine learning, deep learning, and LLM methods. To ensure robustness, each experiment is repeated multiple times and the average results are reported. All methods were fine-tuned using the same 80% training split to ensure fair comparison. For deep learning, we adopt ResNet-18 for icon classification and BERT for permission text classification. CLIP, a multimodal vision-language model, is used to jointly encode app icons and textual descriptions for fraud prediction. To benchmark the effectiveness of the multi-agent design, we also compare against a single-agent GPT-4o baseline, in which GPT-4o acts as a zero-shot classifier and directly performs inference on the 20% test set without training. We adopt four key metrics: Accuracy (ACC for short), Precision, Recall, and F1-Score for the evaluation.

B. Results

As shown in Table I, AgentDroid achieves the highest performance across all evaluation metrics, with an Accuracy of 91.70% and an F1 score of 91.68%. Compared to traditional models such as Drebin and DroidDetector, as well as recent LLM-based models including AppPoet and LAMD, our tool demonstrates substantial improvement. The inferior performance of single-modality baselines such as BERT (text-only) and ResNet-18 (icon-only) highlights the necessity of integrating multimodal features. Notably, GPT-4o used in a single-agent zero-shot setting performs poorly with an F1 score of 80.46%, underscoring the effectiveness of our multi-agent collaborative design. These results confirm the advantage of integrating multimodal analysis with agent specialization for robust fraud detection.

V. Conclusion and Future Work

In this paper, we propose an Android fraudulent application detection tool named AgentDroid, which is based on multimodal analysis and multi-agent collaboration. Our preliminary evaluation demonstrates the effectiveness of AgentDroid. Through multi-agent collaboration and the use of specialized agent tools, our approach achieves higher detection accuracy than existing methods, demonstrating its potential practicality. The dataset and our source code are publicly available at https://github.com/Wwstarry/LLM4Fraud.

In future work, we plan to extend AgentDroid to incorporate dynamic analysis techniques to capture runtime behaviors of applications. We also intend to evaluate the tool on larger and more diverse datasets to further validate its generalizability and robustness.

References

[1] Arp et al. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, volume 14, pages 23–26, 2014.

[2] Anthony Desnos et al. *Androguard Documentation*, 2018. Obtenido de Androguard.

[3] Dong et al. Frauddroid: Automated ad fraud detection for android apps. In *Proc. of ESEC/FSE '18*, pages 257–268, 2018.

[4] Maria Gersen. Mobile ad fraud: Definition, types, detection, 2016. https://clickky.biz/blog/2016/12/mobile-ad-fraud-definition-types-detection/.

[5] He et al. Deep residual learning for image recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[6] Tianliang Lu, Yanhui Du, Li Ouyang, Qiuyu Chen, and Xirui Wang. Android malware detection based on a hybrid deep learning model. *Security and Communication Networks*, 2020(1):8863617, 2020.

[7] Qian et al. Lamd: Context-driven android malware detection and classification with llms. In *2025 IEEE Security and Privacy Workshops (SPW)*, pages 126–136. IEEE, 2025.

[8] Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[9] Hugo Touvron et al. Training data-efficient image transformers & distillation through attention. In *Proc. 38th ICML*, volume 139, pages 10347–10357, 2021.

[10] Yuan et al. Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Science and Technology*, 21(1):114–123, 2016.

[11] Zhao et al. Apppoet: Large language model based android malware detection via multi-view prompt engineering. *Expert Systems with Applications*, 262:125546, 2025.