NanoTG: Automated Fine-Grained Mobile Traffic Generation with LLM GUI Agents

Wentao Wang*†, Yujia Zhu*†, Baiyang Li*†✉, Han Li*, Qingyun Liu*†
*Institute of Information Engineering, Chinese Academy of Sciences
†School of Cyber Security, University of Chinese Academy of Sciences
Email: libaiyang@iie.ac.cn

***Abstract*—Fine-grained network traffic identification is crucial for service quality assurance, security monitoring, and traffic scheduling. However, the construction of fine-grained datasets currently relies heavily on manual efforts, suffering from prohibitive costs, low efficiency, and rapid obsolescence due to frequent application updates. To overcome these scalability bottlenecks, we propose NanoTG, an automated framework leveraging a Plan-Execute-Reflect Agent architecture for reliable mobile traffic generation. NanoTG autonomously explores target apps to identify fine-grained app-activities and achieves semantic-level traffic attribution by aligning network flows with granular execution logs. We evaluated NanoTG on 10 popular applications. During exploration phase, NanoTG uncovered 21.6% more unique IP addresses and 22.7% more SNIs than baseline tools. This demonstrates that NanoTG can identify more fine-grained app-activities. For app-activity quality, NanoTG generated 113 high-quality tasks and achieved a 62% improvement in network relevance score over baseline methods. We further discuss the effectiveness of NanoTG-generated traffic, and preliminarily validate its realism and downstream task utility.**

***Index Terms*—network traffic, GUI agent, application classification, android**

I. Introduction

The proliferation of mobile applications has fundamentally altered global internet usage patterns [1], establishing app-level traffic identification as a critical component of network management and security. In this context, data-driven approaches—ranging from traditional machine learning [2], [3] to state-of-the-art deep learning approaches [4], [5] —have become the mainstream solutions. These methods demonstrate superior capabilities in capturing latent patterns within app traffic data. However, the success of these data-driven models is predicated on the large-scale, high-quality datasets.

Despite the critical need for automation, the construction of mobile traffic datasets has lagged behind the evolution of identification models. Our survey of related work, summarized in Table I, reveals two primary limitations. First, existing datasets suffer from a lack of semantic granularity. The majority of studies focus solely on App-level identification, overlooking the distinct traffic behaviors associated with the complex functionalities of modern applications. Prior research [21] indicates that this lack of semantic information significantly degrades model performance in real-world scenarios. Second,

TABLE I: Survey of Mobile Network Traffic Datasets: Scale and Label Granularity.

Reference	Year	#Apps	Avg. Acts/App	👥
App-level (coarse)				
[6]	2015	5	—	●
[7]	2016	950	—	◐
[8]	2018	12	—	●
[9]	2018	512	—	◐
[10]	2019	40	—	●
[11]	2020	142	—	●
[12]	2020	35	—	●
[13]	2021	101	—	●
[14]	2021	60	—	○
[15]	2024	60	—	●
[16]	2024	350	—	●
App-Activity level (fine)				
[17]	2021	16	2	●
[18]	2021	23	3–4	●
[19]	2022	9	3	●
[20]	2024	20	5	●

Method: ○ fully automated; ◐ manual automation hybrid; ● fully manual. **Summary:** 11 coarse-grained datasets (2,227 apps) vs. 4 fine-grained datasets (68 apps), highlighting the scarcity of activity-level labels.

current collection pipelines require a high degree of manual involvement, which impedes large-scale data generation. This reliance on human intervention leads to prohibitive maintenance costs and poor dataset timeliness. As demonstrated in [22], such staleness in training data further negatively impacts classification accuracy. Consequently, simultaneously achieving automated collection and fine-grained semantic labeling remains a significant challenge.

To bridge this gap, we propose NanoTG, a framework that tightly couples automated app interactions with traffic generation. Unlike previous approaches, NanoTG explicitly associates agent actions with the resulting network requests, thereby enabling the automated generation of high-quality network traffic with precise, fine-grained labels. Specifically, given a target app, NanoTG automatically explores its GUI to discover activities that trigger network communication. To ensure data precision, the system generates a distinct task for each discovered activity. By controlling a single device to execute tasks sequentially, NanoTG isolates the network traffic generated by the target app and leverages detailed logs

✉Baiyang Li is the corresponding author.

ISBN 978-3-903176-82-9 © 2026 IFIP

generated during execution to attribute the traffic to specific operations with fine-grained temporal precision. This approach enables an in-depth analysis of app traffic behavior patterns.

In summary, the main contributions of this paper are as follows:

- To the best of our knowledge, NanoTG is the first framework to introduce GUI agents into the domain of mobile traffic generation. It establishes a fully automated pipeline that significantly minimizes manual labor while ensuring the generated datasets retain rich semantic granularity.
- NanoTG autonomously explores network-centric activities and executes synthesized tasks to trigger authentic traffic. Crucially, it incorporates a semantic alignment mechanism that leverages granular execution logs to map network packets to specific user operations, thereby achieving precise data annotation.
- We evaluate NanoTG on 10 diverse real-world applications. The framework successfully identified 113 tasks, and we further discuss the effectiveness of NanoTG-generated traffic by comparing it with human-generated traces and evaluating its downstream task utility. These results preliminarily validate NanoTG's practical potential for automated fine-grained traffic dataset construction.

The remainder of this paper is organized as follows. Section II introduces the background of traffic generation and GUI-Agents. Section III details the methodology of NanoTG. Section IV evaluates the effectiveness of the proposed framework, Section V discusses limitations, and and Section VI concludes the paper.

II. Background and Related Work

This section analyzes the limitations of existing mobile traffic dataset construction methods and reviews the evolution of GUI automation tools. We aim to highlight the critical gap between current data collection capabilities and the requirements for fine-grained traffic analysis, leading to the specific research challenges addressed in this paper.

A. Dilemma in Dataset Construction

Constructing a mobile traffic dataset involves a fundamental trade-off between *label granularity* and *automation scale*. As summarized in Table I, existing datasets generally fall into two categories:

- Coarse-grained Automated Datasets. To achieve large-scale collection, previous studies [7], [9], [14] utilize UI fuzzers (e.g., Android Monkey) to drive apps stochastically. While these methods significantly reduce human cost, they typically provide only App-level labels (e.g., This flow belongs to "YouTube"). Crucially, the traffic generated by random clicks often lacks the logical sequences and semantic patterns of real user behavior, rendering it unsuitable for training models that require fine-grained activity recognition.
- Fine-grained Manual Datasets. Recent research has shifted towards App-Activity-level identification [19], [23], as recognizing concrete in-app behaviors from network traffic enables finer-grained traffic management and can even facilitate user behavior inference attacks. However, establishing the ground truth for these activity datasets currently relies heavily on manual operation or strictly predefined scripts. This dependency makes such datasets expensive to construct, difficult to update, and limited in coverage. Consequently, there is currently no framework that simultaneously achieves the automation and scalability of stochastic exploration and the semantic granularity of manual collection.

B. Evolution of GUI Automation Tools

The quality of a network traffic dataset is fundamentally determined by the GUI automation driver used to generate interactions. Initially, tools like Monkey [24] and DroidBot [25] relied on stochastic events or heuristic state exploration (e.g., DFS). While efficient for crash testing, these methods produce semantically meaningless sequences, resulting in chaotic network flows that fail to capture logical user workflows. To address this, subsequent research introduced deep learning models to mimic human interaction patterns [26]. Although these approaches improved operational realism by learning from manual trajectories, they remained black-box solutions unable to align specific UI operations with underlying app semantics. Most recently, LLM-based agents like DroidAgent [27] have enabled intent-driven automation, allowing for complex, multi-step task execution. However, existing agents are primarily optimized for software testing objectives—such as maximizing code coverage or finding bugs—rather than traffic collection.

Consequently, directly applying them to dataset construction introduces significant challenges regarding data purity and label accuracy, as their exploration policies are not designed to isolate specific network behaviors.

C. Research Scope and Challenges

The overarching goal of this paper is to leverage GUI agents to simulate human operation, thereby replacing manual effort in generating traffic annotated with semantic tags (App-Activities). To successfully transform a "Testing GUI Agent" into a viable "Traffic Generator", two critical challenges need to be addressed:

- Identification of App Activities. App activities exhibit significant heterogeneity; prior research has predominantly relied on manual identification of target activities prior to traffic collection. Such methodologies, however, necessitate substantial domain expertise in application semantics and prove impractical when scaling to the vast volume of contemporary app ecosystems.
- Reliable traffic generation. Although GUI Agents demonstrate considerable flexibility, their inherent uncertainty in execution outcomes precludes their direct application to network traffic generation. This uncertainty manifests in two distinct dimensions: (1) outcome uncertainty, wherein GUI Agents may fail to complete designated tasks successfully; and (2) process uncertainty, referring to the

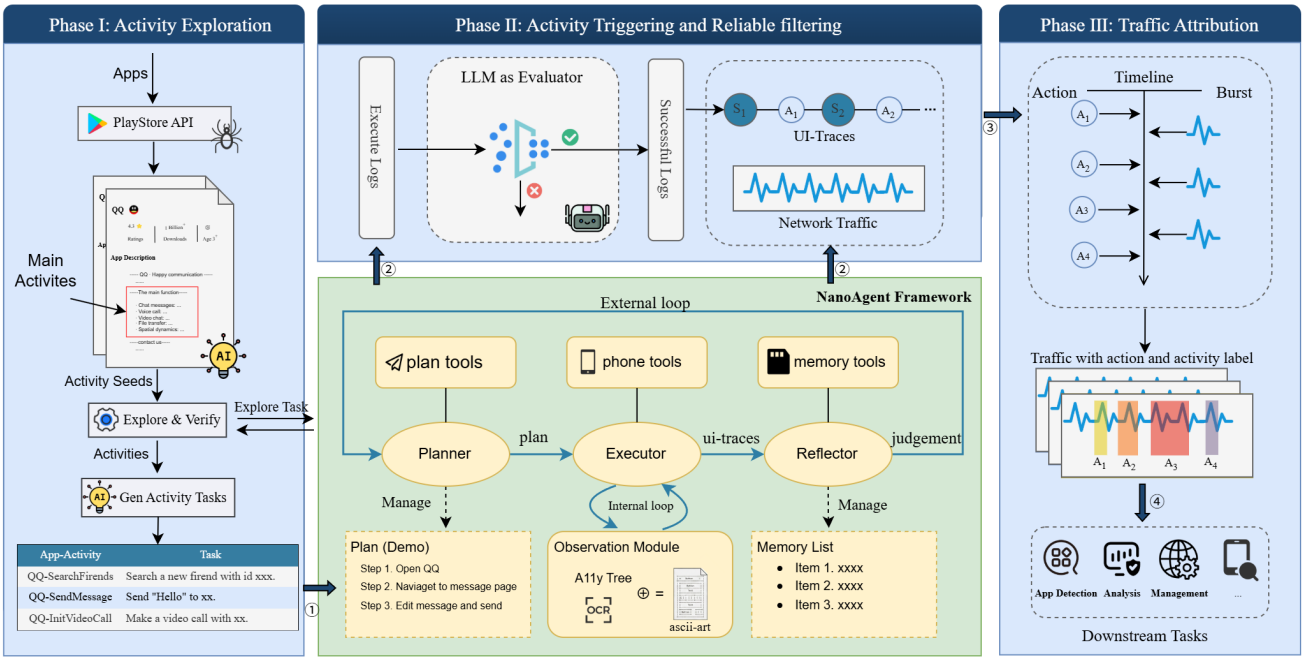


Fig. 1: Overall architecture and workflow of the NanoTG framework.

variability in execution trajectories—different interaction paths adopted by the agent can lead to divergent patterns in the resulting network traffic.

III. Methodology

This section presents the comprehensive workflow of the NanoTG framework, which is systematically organized into three phases as illustrated in Fig. 1. At the core of this workflow lies NanoAgent, a planning-execution-reflection mobile agent capable of performing natural language tasks on smartphones. NanoTG's implementation details are provided in [28].

To address the aforementioned challenges, we decompose the entire pipeline into three distinct phases: Phase I focuses on the identification of app activities, while Phases II and III are dedicated to reliable traffic generation:

1) ***Phase I: Activity Exploration.*** NanoTG automatically explores the target app, identifying primary activities and mapping them into executable task descriptions.
2) ***Phase II: Activity Triggering and Reliability Filtering.*** NanoAgent executes the generated tasks while NanoTG employs an automated verification mechanism to ensure data quality. This phase filters out failed attempts, thereby significantly improving the reliability of the generated dataset.
3) ***Phase III: Fine-Grained Traffic Attribution.*** Leveraging the granular execution logs exported by NanoAgent, NanoTG explicitly aligns network traffic with specific user actions. This step enhances the interpretability of the dataset by attributing traffic flows to precise operations.

Since NanoAgent serves as the foundational execution engine utilized throughout Phase I and Phase II, we first detail its architecture and implementation in the following subsection, followed by a detailed description of each specific phase.

A. NanoAgent Architecture

NanoAgent adopts the *Plan-Execute-Reflect* paradigm [29]. As illustrated in Fig. 1, the *Planner* decomposes the task T into an ordered plan P; the *Executor* sequentially executes each step via atomic UI operations; the *Reflector* validates step completion and updates short-term memory M with execution feedback. This outer-loop cycle iterates until the task is resolved.

Observation Module. The observation module combines structural and visual cues to generate robust observations O_t. Specifically, it parses the Android UI tree—pruning layout-only or invisible nodes—and encodes valid elements into a compact Markdown list. Complementarily, PaddleOCR [30] extracts text from screenshots, discarding low-confidence detections (< 0.8). This dual-source fusion ensures comprehensive perception even when accessibility data is obfuscated in commercial apps. Within each step, the Executor runs an inner loop: it perceives O_t, selects a primitive action (`Click`, `Input`, `Swipe`, `Press`, or `Idle`), and executes it via `uiautomator2`. Upon emitting `Idle`, the Executor outputs the observation–action trajectory. The Reflector then assesses this trajectory to produce textual feedback R_i and enrich M for subsequent planning iterations.

Logging Module. To facilitate rigorous auditing and debugging, the Logging Module maintains a persistent $State$ object that captures the full lifecycle of the agent's operation. Prior to task execution, the module registers essential metadata,

including the task description T, the target app name, and the package identifier. During execution, it sequentially logs the complete interaction history. This includes every prompt sent to the LLM, the model's textual responses, specific tool invocations, and the resulting changes in the UI context (capturing both the Accessibility Tree and screenshots). The logging module supports two distinct export formats to serve different downstream needs:

- *Full Execution Trace:* A verbatim record of every decision step, interface state, and operation. This comprehensive log is optimized for detailed result evaluation by either LLMs or human reviewers.
- *Operational Abstraction:* A synthesized breakdown of the process, detailing how NanoTG decomposed the main goal into sub-tasks and the specific actions performed for each. This format allows the execution logic to be crystallized into reproducible automation scripts for future replay.

B. Phase I: Activity Exploration

The primary objective of this phase is to automatically identify key network-related activities within a target app. While Large Language Models (LLMs) possess vast general knowledge, directly querying them for specific app behaviors often yields hallucinations, particularly for apps with sparse online documentation. To address this challenge, we implement a retrieval-augmented exploration strategy that grounds LLM reasoning in verifiable app metadata.

The process proceeds in three logical steps. First, we retrieve the app's meta-description from the Google Play Store based on its package name. This description, which outlines the app's core functionality, serves as ground truth context. We instruct the LLM to analyze this context and hypothesize a list of potential activities likely to trigger network requests (e.g., "streaming video" or "syncing data"). Second, we construct a directive prompt for NanoTG to empirically verify these hypotheses. Guided by its Planner and Memory mechanisms, NanoTG prioritizes investigating the identified potential activities. Crucially, the agent utilizes its reflection mechanism to prune redundant interaction paths, recording the validated functionalities of visited interfaces into its Memory. Third, we extract the structural information recorded in the Memory and instruct the LLM to synthesize executable task prototypes corresponding to the discovered activities.

C. Phase II: Activity Triggering and Reliability Filtering

In this phase, NanoAgent sequentially executes the tasks generated in Phase I, so that each App-Activity can be triggered independently and its resulting network traffic can be captured in isolation. To ensure the correctness of the triggering process, we introduce an LLM-as-Evaluator module to automatically identify and filter out erroneous executions. The implementation details of this module are presented later in this subsection. This pipeline effectively filters out unsuccessful execution attempts, ensuring that the final dataset comprises only high-fidelity traffic flows associated with successfully completed activities.

Algorithm 1: Burst-based Temporal Proximity Attribution (BTPA)

Input: Packet set $\mathcal{P}$ (time-sorted), Action sequence $\mathcal{S}$, Gap δ_{gap}, Window τ_{win}
Output: Mapping $\mathcal{M} : \mathcal{B} \to \mathcal{S} \cup \{Background\}$

Initialize $\mathcal{B} \leftarrow \emptyset$, active flows map $\mathcal{F}$
foreach $p \in \mathcal{P}$ **do**
  $k \leftarrow$ GetFlowKey(p)
  if $k \in \mathcal{F}$ ***and*** $time(p) - end\_time(\mathcal{F}[k]) > \delta_{gap}$ **then**
    Finalize current burst in $\mathcal{F}[k]$ and add to $\mathcal{B}$
  end
  Append p to current (or new) burst in $\mathcal{F}[k]$
end
Add all active bursts in $\mathcal{F}$ to $\mathcal{B}$
foreach $b \in \mathcal{B}$ **do**
  $t_{start} \leftarrow start\_time(b)$
  $Candidates \leftarrow \{a \in \mathcal{S} \mid 0 \le t_{start} - time(a) \le \tau_{win}\}$
  if $Candidates \neq \emptyset$ **then**
    $a_{best} \leftarrow \arg\min_{a \in Candidates}(t_{start} - time(a))$
    $\mathcal{M}[b] \leftarrow a_{best}$
  else
    $\mathcal{M}[b] \leftarrow Background$
  end
end
return $\mathcal{M}$

LLM-as-Evaluator Module. Verifying whether an agent successfully triggered the intended activity is critical for label accuracy. The Logging Module captures the granular execution history. For each logical step S_i in the plan, the corresponding sequence of observation-action pairs is recorded as a trace segment, denoted as $Trace_i = \{(O_1, A_1), \ldots, (O_m, A_m)\}$. To evaluate the overall success of a task, we aggregate the set of all trace segments $\{Trace_1, \ldots, Trace_n\}$ associated with that task. This aggregated context, combined with the original task description, is fed into an LLM evaluator. The model analyzes the visual and textual evidence to determine whether the user intent was fulfilled, automatically filtering out instances where the agent failed or deviated from the objective.

D. Phase III: Traffic Attribution

To precisely attribute network traffic to specific app operations, we propose the Burst-based Temporal Proximity Attribution (BTPA) algorithm. This method temporally aligns fine-grained operation logs recorded by the Logging Module with resulting network traffic, thereby facilitating the filtering of traffic induced by redundant interactions.

Since standard 5-tuple analysis fails to distinguish operations within persistent flows, we adopt the "traffic burst" as the atomic attribution unit. Sequential packets sharing a 5-tuple are aggregated into a burst b. A burst terminates when the inter-arrival time Δt between consecutive packets exceeds a threshold δ_{gap} (set to 0.5s). This process segments the continuous stream $\mathcal{P}$ into a discrete burst set $\mathcal{B}$. Then, we map each burst $b_j \in \mathcal{B}$ to its causal operation $a_{best} \in \mathcal{S}$ using

TABLE II: Test apps in this paper.

ID	App	Category
QQ	QQ	Social Networking
WC	WeChat	Social Networking
QQM	QQ Music	Music Streaming
BLI	Bilibili	Video Streaming
JD	JD.com	Shopping
ZH	Zhihu	Q&A / Knowledge Community
TM	Tencent Meeting	Conferencing
XHS	Xiaohongshu (RED)	Social / Lifestyle
DS	DeepSeek	LLM Assistant
Qwen	Qwen (Tongyi Qianwen)	LLM Assistant
AW	AndroidWorld(20 apps)	-

the "Nearest Neighbor Priority" principle. A valid attribution must satisfy two constraints:

- Precedence: The operation must precede the burst, i.e., $timestamp(a_i) \leq start(b_j)$.
- Timeliness: The latency must fall within a maximum causal window τ_{win} (5.0s), i.e., $start(b_j) - timestamp(a_i) \leq \tau_{win}$.

The burst is attributed to the operation satisfying these conditions with the minimal time difference. Bursts with no matching operation are classified as *Background Traffic* (e.g., heartbeats). The detailed logic is presented in Algorithm 1.

IV. Experimental Evaluation

This section systematically evaluates the NanoTG in automating the construction of mobile traffic datasets. Our evaluation is guided by the following research questions:

- **RQ1**: Can NanoTG effectively and automatically discover fine-grained app activities?
- **RQ2**: Is NanoTG capable of autonomously and reliably generating high-fidelity network traffic?

A. Experimental Setup

1) Test Apps: Table II details the tested applications selected for this study. Our traffic collection relies on PCAPdroid [31], which can filter out system traffic and capture app traffic directly through Android's VPNService API. To balance task diversity and traffic capture precision, we selected 10 popular Chinese apps from different categories.

2) Baselines: We compare NanoTG against three baselines representing distinct paradigms of GUI automation: *Monkey*, *Droidbot* and *DroidAgent*. These tools serve as representative benchmarks for random, dfs, and intelligent automation approaches, respectively.

3) Device and Environment: All experiments were conducted on a physical Android device (vivo iQOO Neo3, Android API 32) equipped with 12GB RAM and 128GB storage. For consistency, all tools operated under a pre-authenticated state using dedicated test accounts. This setup ensures that the generated traffic reflects valid user interactions rather than repeated authentication flows. Throughout all components that rely on large language models, we consistently employ DeepSeek v3.2 Exp, a representative text-based large language model.

B. Activities Discovery

Metrics. We employ two categories of metrics to evaluate NanoTG's capability in discovering app activities: coverage metrics and the Network Relevance Score (NRS). Coverage is measured by the count of unique IP addresses and Server Name Indications (SNIs). A higher count indicates broader interface exploration and a greater likelihood of discovering diverse activities. To assess semantic quality, we propose the Network Relevance Score (NRS), which quantifies the correlation between discovered activities and actual network behavior. This ensures that dataset construction prioritizes activities with high network relevance.

Activities are classified into four tiers based on their network dependency:

- Composite Operations (Score 3): Complex functionalities with heavy network dependency (e.g., video calls, checkout flows).
- Standard CRUD (Score 2): Typical Create, Read, Update, Delete operations triggering standard request-response patterns.
- Cached Read (Score 1): Read operations likely to hit local caches (e.g., viewing chat history), with uncertain network triggers.
- Network-free (Score 0): Purely local interactions (e.g., theme switching).

For an app with n generated tasks having scores $s_1, s_2, \ldots, s_n$, the normalized NRS is defined as:

$$\text{NRS} = 100 \times \frac{\sum_{i=1}^{n} s_i}{3 \cdot n}$$

This metric balances task quantity with semantic quality, normalizing results to a 0–100 scale for cross-app comparison.

Experimental Setup. Each tool was allocated one hour to explore each app. For Monkey, which lacks native time constraints, we employed a duration-controlled script to approximate this limit. To prevent cross-contamination, app data was fully reset between trials.

Results. Since Monkey and DroidBot operate without semantic awareness and cannot output structured activities, the NRS comparison is limited to DroidAgent. However, for coverage metrics (IPs and SNIs), all tools are compared. As illustrated in Fig. 3, NanoTG demonstrates superior coverage, detecting the highest volume of valid endpoints. It outperforms the nearest baseline by 21.6% in unique IP addresses and 22.7% in SNIs. Ablation studies reveal the critical role of integrating app description metadata: *NanoTG-proto* (operating without external knowledge) performs comparably to DroidAgent, confirming that knowledge-driven exploration significantly enhances the discovery of network-active components. Notably, the traditional DFS-based tool, DroidBot, captures more endpoints than the vanilla agent baseline, suggesting that without semantic guidance, systematic algorithmic traversal can be more efficient than standard agent-based exploration.

Figure 2 presents the quality assessment results. Regarding generation efficiency, DroidAgent produced 180 tasks but suffered from high redundancy (65 duplicates) and failed

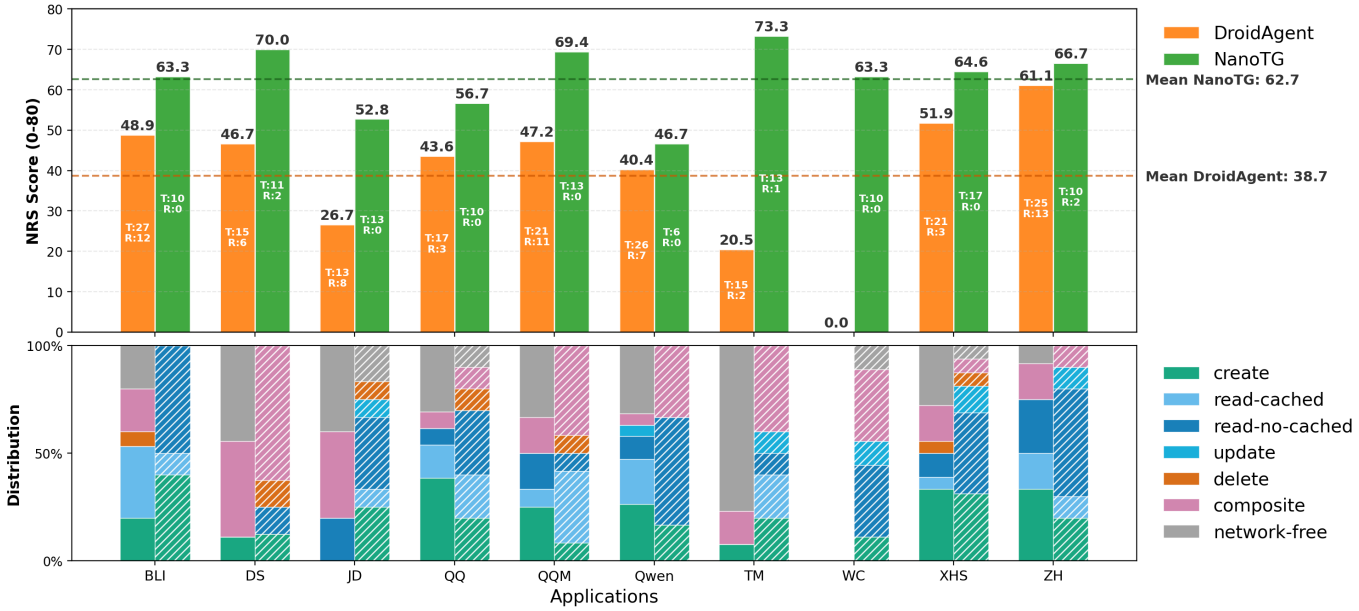


Fig. 2: Comparative analysis of task quality metrics across target applications. The upper panel presents the Network Relevance Scores (NRS) for generated tasks, while the lower panel illustrates the distribution of functional task types.

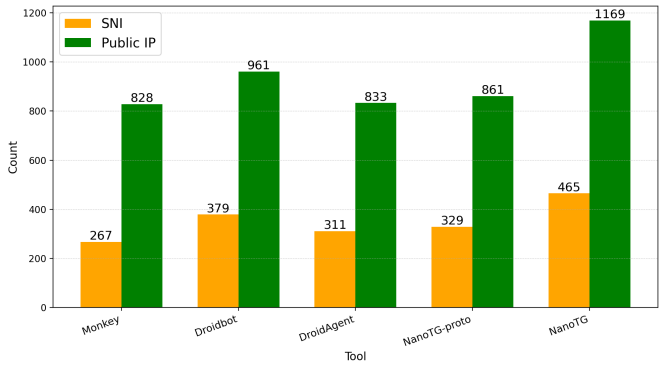


Fig. 3: Network Coverage Comparison: Number of Public IP and SNI Detections per Tool.

completely on WeChat due to execution errors. In contrast, NanoTG generated 113 tasks with negligible redundancy (only 5 duplicates). This efficiency stems from our architecture, which synthesizes tasks post-exploration and explicitly constrains the LLM to filter duplicates. NanoTG consistently achieves higher NRS scores than DroidAgent across all tested apps, with an average relative improvement of 62% (62.7 vs. 38.7). The distribution analysis (bottom section) confirms that NanoTG's tasks are predominantly network-intensive. Conversely, DroidAgent generates a significant proportion of network-free tasks (e.g., pure UI navigation), likely because its exploration strategy maximizes page coverage rather than network relevance.

C. Traffic Generation

Metrics. We evaluate the reliability of NanoTG's traffic generation from two primary perspectives: Evaluation Accuracy and Traffic Fidelity. To ensure a comprehensive assessment, we employ three specific metrics:

- Execution Success Rate (ESR): Quantifies the agent's robustness in interpreting instructions and navigating UIs.
- Evaluation Accuracy: Measures the precision of the LLM-as-evaluator in verifying task completion, critical for filtering failed traces.
- Traffic-Operation Alignment: Evaluates the temporal correspondence between generated traffic bursts and specific UI operations using the BTPA algorithm.

Experimental Setup. To enhance experimental reliability, we expanded the evaluation scope to comprehensively assess the LLM-as-evaluator module. We incorporated the *AndroidWorld* [32] benchmark, which includes 20 apps and 116 tasks of varying difficulty levels. Since these benchmark apps are operated in an offline context, traffic-operation alignment analysis is not performed for this subset. For each task, the application is fully terminated after each execution to prevent state contamination across trials.

Results. We analyze the results from the following two perspectives.

a) Execution and Evaluation Accuracy: We first analyze the overall success rate of task execution and the LLM-as-evaluator module, as shown in Table III. NanoTG achieved an execution success rate exceeding 60% on both the AndroidWorld benchmark and its own generated tasks. The overall automated evaluation accuracy surpassed 80% (85.0% on AndroidWorld and 83.2% on NanoTG tasks), indicating that

the system can effectively detect erroneous executions and prevent noise injection into the downstream dataset.

To further analyze evaluation effectiveness, we manually investigated the causes of execution failures (Table IV) and identified scenarios where the evaluator is prone to *False Positives* (misclassifying failed executions as successful). We categorize failure causes into three types:

- *Perceptual & Capability Constraints (Ability):* Failures stemming from observation limitations. For example, in the task *"AddChatWithDeepThinking,"* the toggle is color-coded without text, making its state invisible to the accessibility tree.
- *Environmental Volatility (Env):* External disruptions, such as expired meeting IDs or strict anti-bot mechanisms (e.g., CAPTCHAs).
- *Instruction Alignment (Other):* Ambiguous task descriptions leading to hallucinated paths.

Manual analysis reveals that the *Ability* category is most prone to evaluation errors. Since the evaluator relies on text-based logs, information loss in the accessibility tree can lead to inaccurate judgments. Conversely, the *Env* category is least prone to error, as environmental failures typically exhibit clear signals (e.g., repeated verification challenges). The primary error mode for the evaluator is *False Positive* bias. For instance, in an *"AddItemToShoppingCart"* task, if the agent fails to click the final button but remains on the product page, the visual context mimics a success state, misleading the evaluator. Future iterations could mitigate this by enforcing stricter verification prompts.

b) Traffic-Operation Alignment: We evaluate the precision of the Burst-based Temporal Proximity Attribution (BTPA) algorithm by visualizing network traffic I/O patterns. We categorize nine distinct activities into complexity tiers: Group A (Complex Multi-step Tasks) and Group B (Streaming Media Tasks). The comparative attribution results are illustrated in Fig. 4.

In Group A, the method successfully identifies key semantic operations. As evidenced in Fig. 4, distinct traffic bursts (e.g., light purple/pink segments in Fig. 4a) are accurately aligned with their causal actions. However, high operation density can introduce minor attribution drifts due to interleaving network-silent UI interactions.

In Group B, performance is contingent on interaction concurrency. In straightforward streaming contexts (e.g., Fig. 4d), attribution remains robust. Conversely, concurrent operations during continuous playback (e.g., Fig. 4e and Fig. 4f) can obscure temporal boundaries, resulting in occasional misattribution. Overall, BTPA remains highly effective for standard interactions and successfully isolates core functional traffic even in complex scenarios.

V. Discussion and Limitations

A. Discussion

Similarity to human-generated traffic. To directly compare NanoTG with manual collection, we collected 100 pcap traces across 10 reproducible tasks and applied Dynamic Time Warping (DTW) to packet-length time series sampled at 200ms. As shown in Figure 5, the average normalized similarity between human-generated and NanoTG-generated traces reaches 74.7%, and deterministic tasks obtain scores above 0.8. This indicates that NanoTG can reproduce stable user workflows and generate traffic patterns close to manual collection, while dynamic tasks still introduce expected variance due to pop-ups, content refreshes, and streaming states.

Utility in downstream classification. We further evaluate whether NanoTG-generated traffic contains sufficient discriminative information for practical traffic analysis. We construct two datasets, namely *manual-collected* and *agent-generated*, each covering 10 activity categories. Flow-level features are extracted using CICFlowMeter, where sensitive identifiers (e.g., IP addresses, ports) are removed to prevent shortcut learning, while temporal statistical features are retained. We evaluate five classical classifiers, as shown in Table V. The *manual-collected* dataset consistently achieves the best performance, while the *agent-generated* dataset still attains a competitive average F1 score of 0.5352, indicating that NanoTG-generated traffic preserves meaningful class-discriminative patterns and can effectively support downstream traffic classification.

TABLE III: Evaluation of Agent Execution Reliability and Auto-Evaluator Precision.

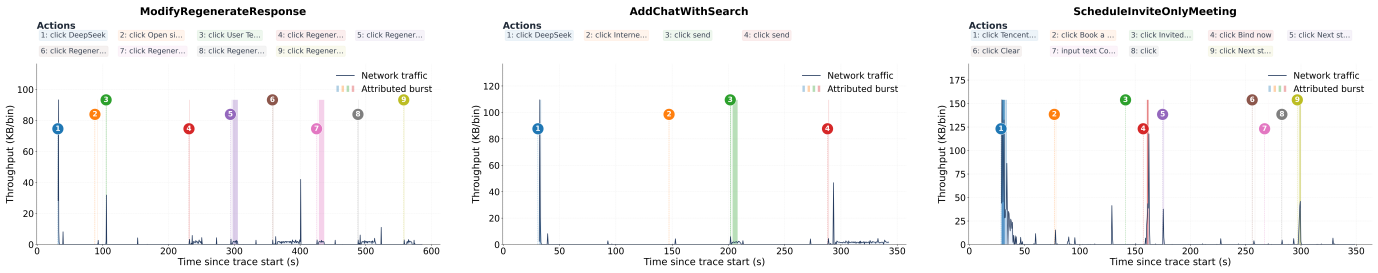
Task Source	Execution Success Rate	Auto-Eval Accuracy
AndroidWorld	60.3%	85.0%
NanoTG Task	68.1%	83.2%

TABLE IV: Granular Breakdown of Task Execution Results and Failure Analysis per Target Application.

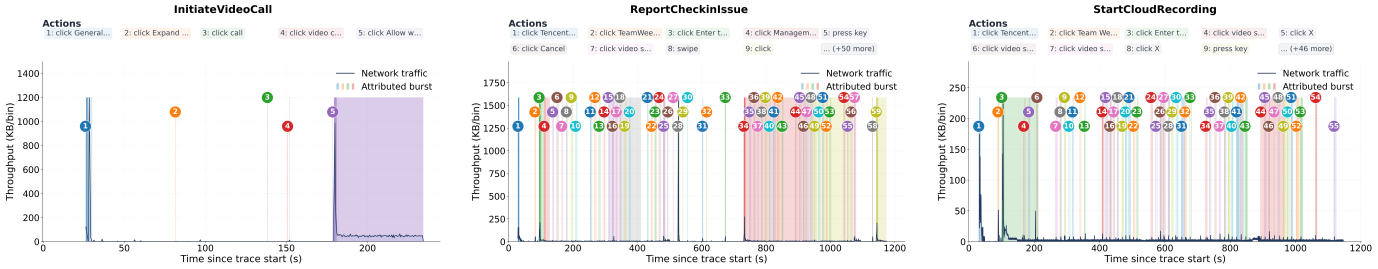
APP ID	Total	Success	Fail	Fail Reasons		
				Ability	Env	Other
WC	10	6	4	0	1	3
BLI	10	9	1	1	0	0
QQM	13	13	0	0	0	0
QWen	6	5	1	0	0	1
ZH	10	8	2	2	0	0
DS	11	3	8	8	0	0
QQ	10	7	3	0	0	3
JD	13	5	8	0	8	0
XHS	17	12	5	1	2	2
TM	13	9	4	2	1	1
AW	116	70	46	13	3	30
Total	**229**	**147**	**82**	**27**	**15**	**40**

B. Limitations and Future Work

This study is limited in scale and app coverage. Our evaluation covers 10 popular Chinese apps and 10 reproducible tasks for downstream usability analysis, which is not sufficient to fully characterize NanoTG across broader app ecosystems, regions, app markets, network environments, and app versions. In addition, the Network Relevance Score (NRS) is still a subjective indicator: although we define explicit scoring tiers



(a) A-1 (b) A-2 (c) A-3



(d) B-1 (e) B-2 (f) B-3

Fig. 4: The first row represents relatively complex tasks, recorded as Group A; the second row represents tasks involving video/audio streams, recorded as Group B.

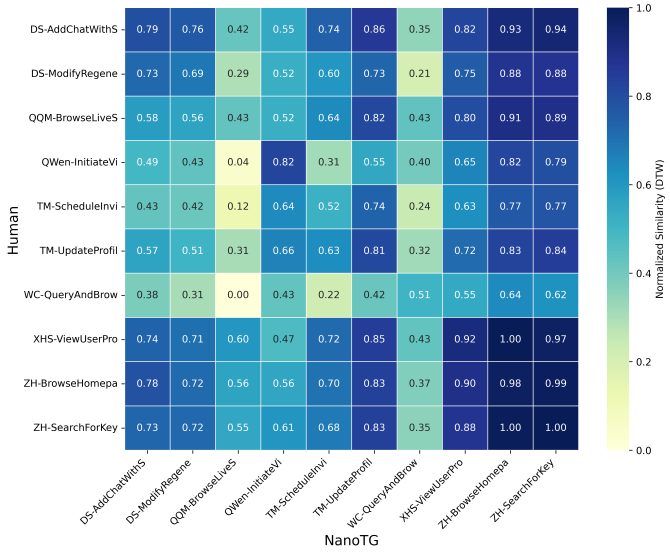


Fig. 5: Heatmap illustrating the normalized DTW similarity between traffic traces generated by humans and NanoTG across different tasks.

to make the evaluation as consistent and fair as possible, assigning semantic activities to these tiers inevitably involves human judgment.

In future work, we will evaluate NanoTG on a larger and more diverse set of apps and construct a large-scale real-world traffic dataset generated by NanoTG. We will also validate its utility under stronger downstream classification settings, so as to further examine the generalizability of GUI-agent-based traffic generation for fine-grained mobile traffic analysis.

TABLE V: Downstream classification performance across manual-collected and agent-generated datasets.

Model	manual-collected (%)				agent-generated (%)			
	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
RF	80.14	76.22	75.08	74.77	78.26	69.88	64.88	66.75
GB	78.61	75.20	73.80	73.96	74.27	66.09	59.27	61.76
KNN	72.12	65.01	65.18	64.01	71.16	58.76	57.57	57.76
SVM-RBF	62.34	59.39	61.03	58.51	52.14	42.94	47.90	43.48
LR	51.03	50.18	52.77	49.44	45.25	37.32	42.91	37.85

Abbreviations: RF=random forest, GB=gradient boosting, LR=logistic regression.

VI. Conclusion

In this paper, we present NanoTG, an automated Android app traffic generation framework designed to replace manual data collection. Leveraging an LLM-based agent, NanoTG autonomously discovers fine-grained, network-centric activities and achieves precise action-level traffic attribution. Comprehensive evaluations demonstrate that NanoTG significantly outperforms existing tools, improving network activity discovery by over 21% and achieving a 62% improvement in task quality over the baseline agent, while maintaining an automatic evaluation accuracy of over 80%. Furthermore, we validate the effectiveness, realism, and downstream utility of the generated traffic. Our results establish the feasibility of utilizing GUI agents to construct fine-grained mobile traffic datasets without manual annotation, highlighting the potential of agent-driven approaches to enable scalable and adaptive traffic analysis in dynamic mobile environments.

REFERENCES

[1] B. Biblow. Mobile app download statistics & usage statistics (2025). [Online]. Available: https://buildfire.com/app-statistics/

[2] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 439–454. [Online]. Available: https://ieeexplore.ieee.org/document/7467370/

[3] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescape, "Traffic classification of mobile apps through multi-classification," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6.

[4] M. H. Pathmaperuma, Y. Rahulamathavan, S. Dogan, and A. Kondoz, "CNN for user activity detection using encrypted in-app mobile data," vol. 14, no. 2, p. 67. [Online]. Available: https://www.mdpi.com/1999-5903/14/2/67

[5] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proceedings of the ACM Web Conference 2022*. ACM, pp. 633–642. [Online]. Available: https://dl.acm.org/doi/10.1145/3485447.3512217

[6] S. Mongkolluksamee, V. Visoottiviseth, and K. Fukuda, "Enhancing the performance of mobile traffic identification with communication patterns," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2, pp. 336–345, ISSN: 0730-3157. [Online]. Available: https://ieeexplore.ieee.org/document/7273638/

[7] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "ReCon: Revealing and controlling PII leaks in mobile network traffic," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, pp. 361–374. [Online]. Available: https://dl.acm.org/doi/10.1145/2906388.2906392

[8] R. Wang, Z. Liu, Y. Cai, D. Tang, J. Yang, and Z. Yang, "Benchmark data for mobile app traffic research," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. ACM, pp. 402–411. [Online]. Available: https://dl.acm.org/doi/10.1145/3286978.3287000

[9] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez, "Bug fixes, improvements, ... and privacy leaks - a longitudinal study of PII leaks across android app versions," in *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_05B-2_Ren_paper.pdf

[10] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "MIRAGE: Mobile-app traffic capture and ground-truth creation," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, pp. 1–8.

[11] X. Wang, S. Chen, and J. Su, "Real network traffic collection and deep learning for mobile app identification," vol. 2020, no. 1, p. 4707909, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2020/4707909. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2020/4707909

[12] T. Van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. Van Steen, and A. Peter, "FlowPrint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Proceedings 2020 Network and Distributed System Security Symposium*. Internet Society. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24412.pdf

[13] T.-D. Pham, T.-L. Ho, T. Truong-Huu, T.-D. Cao, and H.-L. Truong, "MAppGraph: Mobile-app classification on encrypted network traffic using deep graph convolution neural networks," in *Annual Computer Security Applications Conference*. ACM, pp. 1025–1038. [Online]. Available: https://dl.acm.org/doi/10.1145/3485832.3485925

[14] J. Shi, M. Liu, C. Hou, M. Jiang, and G. Xiong, "Online encrypted mobile application traffic classification at the early stage: Challenges, evaluation criteria, comparison methods," in *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, pp. 1128–1135. [Online]. Available: https://ieeexplore.ieee.org/document/9449097/

[15] M. Bayat, J. Garshasbi, M. Mehdizadeh, N. Nozari, A. Rezaei Khesal, M. Dokhaei, and M. Teimouri, "ITC-net-blend-60: a comprehensive dataset for robust network traffic classification in diverse environments," vol. 17, no. 1, p. 165. [Online]. Available: https://bmcresnotes.biomedcentral.com/articles/10.1186/s13104-024-06817-5

[16] S. Zhao, S. Chen, F. Wang, Z. Wei, J. Zhong, and J. Liang, "A large-scale mobile traffic dataset for mobile application identification," vol. 67, no. 4, pp. 1501–1513. [Online]. Available: https://academic.oup.com/comjnl/article/67/4/1501/7235535

[17] Y. Heng, V. Chandrasekhar, and J. G. Andrews, "UTMobileNetTraffic2021: A labeled public network traffic dataset," vol. 3, no. 3, pp. 156–160. [Online]. Available: https://ieeexplore.ieee.org/document/9490678/

[18] Z. Gu, G. Gou, C. Hou, G. Xiong, and Z. Li, "LFETT2021: A large-scale fine-grained encrypted tunnel traffic dataset," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 240–249, ISSN: 2324-9013. [Online]. Available: https://ieeexplore.ieee.org/document/9724372/

[19] I. Guarino, G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapè, "Contextual counters and multimodal deep learning for activity-level traffic classification of mobile communication apps during COVID-19 pandemic," vol. 219, p. 109452. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128622004868

[20] I. Guarino, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mirage-app×act-2024: A novel dataset for mobile app and activity traffic analysis," in *2024 20th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, pp. 663–666. [Online]. Available: https://ieeexplore.ieee.org/document/10770459/

[21] C. Song, D. D. M. Mekala, H. Wang, and R. Martin, "Redefining website fingerprinting attacks with multiagent LLMs," version Number: 1. [Online]. Available: https://arxiv.org/abs/2509.12462

[22] N. Wickramasinghe, A. Shaghaghi, G. Tsudik, and S. Jha, "SoK: Decoding the enigma of encrypted network traffic classifiers," in *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 1825–1843, ISSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/document/11023502/

[23] D. Li, W. Li, X. Wang, C.-T. Nguyen, and S. Lu, "ActiveTracker: Uncovering the trajectory of app activities over encrypted internet traffic streams," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–9, ISSN: 2155-5494. [Online]. Available: https://ieeexplore.ieee.org/document/8824928/

[24] UI/application exerciser monkey | android studio. [Online]. Available: https://developer.android.com/studio/test/other-testing-tools/monkey?hl=zh-cn

[25] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen, "DroidBot: a lightweight UI-guided test input generator for android," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, pp. 23–26. [Online]. Available: http://ieeexplore.ieee.org/document/7965248/

[26] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Humanoid: A deep learning-based approach to automated black-box android app testing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1070–1073, ISSN: 2643-1572. [Online]. Available: https://ieeexplore.ieee.org/document/8952324/

[27] J. Yoon, R. Feldt, and S. Yoo, "Intent-driven mobile GUI testing with autonomous large language model agents," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 129–139, ISSN: 2159-4848. [Online]. Available: https://ieeexplore.ieee.org/document/10638557/

[28] Wentao Wang and Zhu, Yujia and Li, Baiyang and Li, Han and Liu, Qingyun, "NanoTG: Source Code," https://github.com/bulalalla/NanoTG, 2025, accessed: 2025-12-23.

[29] J. Wang, H. Xu, H. Jia, X. Zhang, M. Yan, W. Shen, J. Zhang, F. Huang, and J. Sang, "Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration." [Online]. Available: http://arxiv.org/abs/2406.01014

[30] C. Cui, T. Sun, S. Liang, T. Gao, Z. Zhang, J. Liu, X. Wang, C. Zhou, H. Liu, M. Lin, Y. Zhang, Y. Zhang, H. Zheng, J. Zhang, J. Zhang, Y. Liu, D. Yu, and Y. Ma. PaddleOCR-VL: Boosting multilingual document parsing via a 0.9b ultra-compact vision-language model. [Online]. Available: https://arxiv.org/abs/2510.14528v4

[31] E. Faranda, "emanuele-f/PCAPdroid," original-date: 2020-01-06T18:11:27Z. [Online]. Available: https://github.com/emanuele-f/PCAPdroid

[32] C. Rawles, S. Clinckemaillie, Y. Chang, J. Waltz, G. Lau, M. Fair, A. Li, W. Bishop, W. Li, F. Campbell-Ajala, D. Toyama, R. Berry, D. Tyamagundlu, T. Lillicrap, and O. Riva, "AndroidWorld: A dynamic benchmarking environment for autonomous agents." [Online]. Available: http://arxiv.org/abs/2405.14573