Practical Escape of Exploration Tarpits for Mini-Game Testing in an Industrial Setting

Yuan Cao
Key Lab of HCST (PKU), MOE; SCS
Peking University
Beijing, China
cao_yuan21@stu.pku.edu.cn

Dezhi Ran*
Key Lab of HCST (PKU), MOE; SCS
Peking University
Beijing, China
dezhiran@pku.edu.cn

Haochuan Lu
Tencent Inc.
Shenzhen, China
hudsonhclu@tencent.com

Chao Guo
Tencent Inc.
Shenzhen, China
willliamguo@tencent.com

Xuran Hao
Peking University
Beijing, China
hxr12138@stu.pku.edu.cn

Zhuoru Chen
Capital Normal University
Beijing, China
zdhwydtkr@gmail.com

Ting Xiong
Tencent Inc.
Shenzhen, China
candyxiong@tencent.com

Yuetang Deng
Tencent Inc.
Shenzhen, China
yuetangdeng@tencent.com

Tao Xie*
Key Lab of HCST (PKU), MOE; SCS
Peking University
Beijing, China
taoxie@pku.edu.cn

***Abstract*—Attracting over one billion registered users globally, WeChat's mini-game platform has become one of the largest gaming platforms with hundreds of thousands of published mini-games. To ensure the quality of experiences across a massive number of mini-games, automated UI testing has become essential for WeChat. However, sliding-gesture-induced exploration tarpits, states where a testing tool becomes trapped in repetitive, unsuccessful gesture attempts, cause the testing tool to waste up to 98% of its testing budget due to its inability to execute proper sliding gestures. While mini-games typically contain visual hints (e.g., sliding indicators) guiding the desired sliding gestures, exploiting these hints to escape exploration tarpits faces two major challenges in industrial settings: (1) robustness challenge when exploiting hints from only several discontinuous screenshots, and (2) efficiency challenge to support thousands of concurrent testing services with minimal overhead and costs.**

To address the preceding challenges, we report our experiences in developing and deploying SLIDESCOUT, a three-stage approach for detecting and escaping sliding-gesture-induced exploration tarpits via efficient exploitation of visual hints. First, SLIDESCOUT concurrently monitors the testing progress and detects sliding indicators alongside screenshot collection, improving efficiency by reusing preprocessed results in subsequent stages. Second, SLIDESCOUT reconstructs potential sliding trajectories using multiple heuristics, addressing robustness challenges when precise trajectories are unavailable due to discontinuous screenshots. Third, SLIDESCOUT applies the inferred sliding gestures until it successfully escapes the tarpit, enabling easy integration with existing testing tools. Deployed at WeChat internally for six months, SLIDESCOUT has helped reveal 25,000 crashes and 120,000 JavaScript errors, detecting 50% more crashes compared to the pre-deployment baseline within the same time period. We summarize three major lessons learned from developing and deploying SLIDESCOUT.

*Corresponding authors.

***Index Terms*—GUI Testing, Mini-Apps, Game Testing, Visual Testing, Image Processing, Object Tracking, Exploration Tarpits**

I. INTRODUCTION

Mini-games [1] are lightweight games embedded within super-apps [2] such as WeChat, Facebook, and TikTok. By eliminating download barriers and enabling instant access, mini-games offer seamless gaming experiences for users to enjoy games without leaving their favorite social or messaging apps [3]. Given the unprecedented convenience, mini-games have now attracted over one billion players globally, with two-thirds engaging daily or several times weekly [4], [5]. By the end of 2024, China's mini-game market has generated revenue of approximately 40 billion Chinese Yuan and is expected to reach 60 billion Chinese Yuan in 2025 [6]. Within this market, WeChat, as one of the largest mini-game platforms, has hosted more than one hundred thousand mini-games [7].

Given the massive scale of mini-games and the platform's rapid growth, WeChat has chosen to ensure the quality of user experiences on mini-games with automated visual UI testing [8], [7] for two reasons. First, manual testing is both tedious and costly [9], [10], [11] for the scale of over one hundred thousand mini-games with frequent updates. Second, unlike conventional mobile or web applications, mini-games employ custom game engines with non-standard UI elements [8] that are inaccessible to traditional testing tools, necessitating image-based testing [8] and random exploration strategies [7].

While automated mini-game testing has been deployed at

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00258

WeChat for years [7], we find that a testing tool frequently encounters sliding-gesture-induced "exploration tarpits" [12], i.e., states where testing tool becomes stuck, repeatedly attempting unsuccessful interactions due to the testing tool's inability to generate required *sliding gestures* [13]. As shown in Figure 1, in a one-hour testing of a highly popular mini-game "Earth Journey", a testing tool [7] wastes up to 98% of the testing budget repeatedly attempting unsuccessful interactions, unable to progress beyond certain game states that require specific sliding gestures to advance. While the testing tool can generate random sliding gestures, combinatorial explosion of sliding-gesture space makes random exploration approaches highly ineffective, as the probability of randomly generating the correct sliding gesture is vanishingly small.

While exploration tarpits have been studied and tackled in general mobile app testing [12], [14], existing solutions cannot be migrated to address sliding-gesture-induced exploration tarpits unique in mini-games. VET [12] uses trace analysis to block UI gestures leading to exploration tarpits, while we must pass through the tarpits in mini-games rather than avoiding them. AURORA [14] assumes a finite set of common patterns for mobile app tarpits and learns from known tarpits, but sliding-gesture tarpits in mini-games exhibit highly diverse and game-specific behaviors. Moreover, unlike simple taps, which have a discrete set of possible locations, sliding gestures involve a combinatorial explosion of parameters, making it difficult to learn how to determine starting points, trajectories, velocities, and ending positions.

To understand the unique characteristics of sliding-gesture-induced exploration tarpits, we conduct a motivating study and discover that *visual hints*, provided by developers to guide users toward correct sliding gestures, provide crucial information about required sliding gestures. Specifically, we annotate a dataset SLIDEQUIZ consisting of 200 exploration tarpits from 59 highly popular mini-games from the WeChat testing platform. After inspecting SLIDEQUIZ, we find that such visual hints exist in almost all exploration tarpits with 96% of them featuring sliding indicators, which move along the desired path to indicate the required sliding gesture, such as the pulsating hand icon illustrated in Figure 1. These sliding indicators provide intuitive guidance for users to understand the required sliding direction and distance.

While sliding indicators provide cues to escape tarpits in mini-game testing, building a practical automated sliding-gesture generation tool faces two major industrial requirements for large-scale deployment.

Robustness requirement when exploiting hints from only several discontinuous screenshots. Industrial testing environments typically provide limited, non-continuous screenshots rather than real-time video feeds, making it difficult to reliably detect and interpret sliding indicators that vary dramatically across games in appearance, animation style, and subtlety. Moreover, the tool must accurately translate these visual hints into precise sliding gestures with appropriate parameters that satisfy each game's specific physics requirements.

Efficiency requirement to support thousands of concurrent testing services with minimal overhead and costs. The tool must process visual hints and generate sliding gestures rapidly enough to avoid becoming a bottleneck in high-throughput testing pipelines, while maintaining low computational and memory footprints to enable cost-effective deployment across large-scale testing infrastructures.

To address the preceding industrial requirements for practical tarpit escape in mini-game testing, in this paper, we report our experiences in developing and deploying SLIDESCOUT, a three-stage approach for detecting and escaping sliding-gesture-induced exploration tarpits via robust and efficient exploitation of noisy visual hints to guide proper sliding gestures. In the first stage, SLIDESCOUT monitors the testing progress and concurrently detects sliding indicators alongside screenshot collection, significantly improving inference efficiency by immediately reusing the preprocessed results in subsequent stages. In the second stage, SLIDESCOUT reconstructs potential sliding trajectories using multiple heuristics, addressing the robustness challenge when precise moving trajectories are unavailable due to discontinuous screenshot collection. In the third stage, SLIDESCOUT applies the inferred sliding gestures until successfully escaping the tarpit and returning control to the testing tool, thus enabling easy integration with any testing tool to enhance testing effectiveness.

To assess the efficacy of SLIDESCOUT, we evaluate SLIDESCOUT against Vision-Language Models (VLMs) and heuristic-based baselines. Evaluation results show that SLIDESCOUT successfully escapes 64.5% of tarpits in SLIDEQUIZ, outperforming the best baseline approaches (including heuristics and VLMs) by 650%. Notably, even state-of-the-art VLMs demonstrate limited success in escaping tarpits, failing to generate contextually appropriate sliding gestures grounded in the provided screenshots. When integrating SLIDESCOUT with the WeChat mini-game platform's existing testing tool [7] on 28 popular mini-games, SLIDESCOUT incurs less than 0.1-second overhead while enabling the testing tool to cover 42.08% more unique game states on average.

We have deployed SLIDESCOUT internally at WeChat, a highly popular app with over **one billion** monthly active users. Since January 2025, SLIDESCOUT has served as the default strategy for exploration tarpit handling for all mini-game testing tasks at WeChat. During the six-month deployment, SLIDESCOUT has been invoked more than 500,000 times and has revealed more than 25,000 game crashes and 120,000 JavaScript errors. Compared with the pre-deployment baselines in the same time period, SLIDESCOUT improves the effectiveness of crash detection and JavaScript-error detection by 50% and 17%, respectively.

In summary, this paper makes the following major contributions:

- SLIDESCOUT [15], the first automated and cost-effective approach for escaping exploration tarpits via robust and efficient exploitation of noisy visual hints in mini-game testing.
- SLIDEQUIZ [15], the first annotated dataset of 200 sliding-gesture-induced exploration tarpits in mini-games,

enabling future research in this direction.
- Comprehensive evaluations demonstrating the effectiveness of SLIDESCOUT through controlled experiments and large-scale deployment at WeChat, serving the entire testing platform for 500,000+ times and detecting 145,000+ issues in six months.
- Practical insights and lessons learned from developing and deploying SLIDESCOUT at WeChat.

II. ILLUSTRATIVE EXAMPLES

In this section, we present a real-world example of sliding-gesture-induced exploration tarpits [12] when testing a mini-game "Earth Journey" on the WeChat testing platform, and our insight to escape the tarpits.

As illustrated in Figure 1, when testing the game "Earth Journey", we observe a severe exploration tarpit where the testing tool explores only three distinct game screens in one hour and wastes over 98% of the testing budget while being trapped in a single game screen. Progressing the mini-game requires a specific sliding gesture (dragging a dish to a customer's "idea cloud"), which the tool cannot perform effectively. This scenario extends the concept of exploration tarpit [12] to mini-games, where sliding gestures become the primary barrier to exploration rather than simple UI navigation issues. Unlike traditional mobile apps where tarpits often involve repetitive taps or navigation loops, mini-game tarpits require precise sliding gestures with specific trajectories and timing, rendering existing tarpit handling strategies [12], [14] ineffective.

While the testing tool can randomly generate sliding gestures [7], the probability of randomly producing the correct sliding gesture is prohibitively low. For tapping gestures, the size of the gesture space is proportional to the number of distinct tap locations on the screen, which we denote as n. In contrast, sliding gestures require both a starting position and an ending position, resulting in a gesture space of size $O(n^2)$. More precisely, based on the input generation strategy [7], generating a sliding gesture that precisely starts from the dish and ends at the customer's "idea cloud" in Figure 1 takes over 100 minutes in expectation, being unacceptable for WeChat's large-scale testing requirements.

Insight: Leverage developer guides to escape tarpits. The sliding-gesture-induced exploration tarpits not only pose challenges for testing tools but also mini-game players. Given that most mini-game players have limited engagement time and patience, developers must help users quickly understand gameplay mechanics. Consequently, developers frequently implement visual hints in various forms, such as indicators moving along sliding paths or highlighting the endpoints of sliding paths. In the example shown in Figure 1, an animated hand icon repeatedly moves from the dish to the customer, clearly indicating the required sliding gesture. These visual hints serve as valuable cues for automatically inferring proper sliding gestures to escape exploration tarpits.

III. MOTIVATING STUDY

In this section, we conduct a motivating study on sliding-gesture-induced exploration tarpits and how game developers' visual hints help users escape these tarpits.

A. Construction of SLIDEQUIZ

Given the absence of standardized datasets for investigating and evaluating sliding-gesture-induced exploration tarpits in mini-games, we construct SLIDEQUIZ, the first dataset consisting of 200 exploration tarpits from 59 highly popular commercial mini-games with escape gesture annotations by expert mini-game testers. SLIDEQUIZ is publicly available [15] to facilitate future research in this direction.

The construction methodology of SLIDEQUIZ follows a three-stage process to ensure representativeness. First, we run a state-of-the-practice mini-game testing tool [7] on WeChat's mini-game platform, identifying mini-games where automated exploration achieves limited coverage, suggesting potential exploration tarpits. This filtering process yields 59 mini-games containing at least one exploration tarpit. Second, we cooperate with five experienced engineers of WeChat. These WeChat engineers have a combined total of over 40 years of experience in testing mobile apps. We inspect testing logs of the 59 mini-games, identifying game scenarios indeed requiring sliding-gesture execution to escape the tarpit. To faithfully simulate real-world testing environments where continuous screen recording is impractical due to performance and storage constraints, we collect discontinuous screen capture sequences for each mini-game across multiple devices with varying screen resolutions (ranging from 720×1280 to 1440×2960 pixels), aspect ratios, and display characteristics. Third, the WeChat engineers help mark the precise bounding boxes for both the starting position and ending position of each required sliding gesture to escape tarpits. Through the preceding systematic process, we obtain SLIDEQUIZ with 200 unique exploration tarpits (each containing a sequence of 8-15 screen captures) with escape annotations.

In addition to sliding-gesture annotations for escaping the exploration tarpits, we collaborate with experienced testing engineers to develop a taxonomy of visual hints, categorizing them into four types: (1) animated indicators that move along required sliding paths, (2) highlighted pathways displayed with brighter colors, (3) endpoint highlighting of starting and ending positions of the required sliding-gesture, and (4) textual instructions describing required gestures. We apply the taxonomy to categorize all visual hints in SLIDEQUIZ. For tarpits containing multiple hint types, each type is counted individually.

B. Study Results

Figure 2 presents the distribution of different types of visual hints across the 200 exploration tarpits in SLIDEQUIZ. From the results, we derive two key observations. First, all exploration tarpits in our dataset contain at least one type of visual hint, confirming that game developers consistently provide guidance mechanisms to help users escape tarpits. This

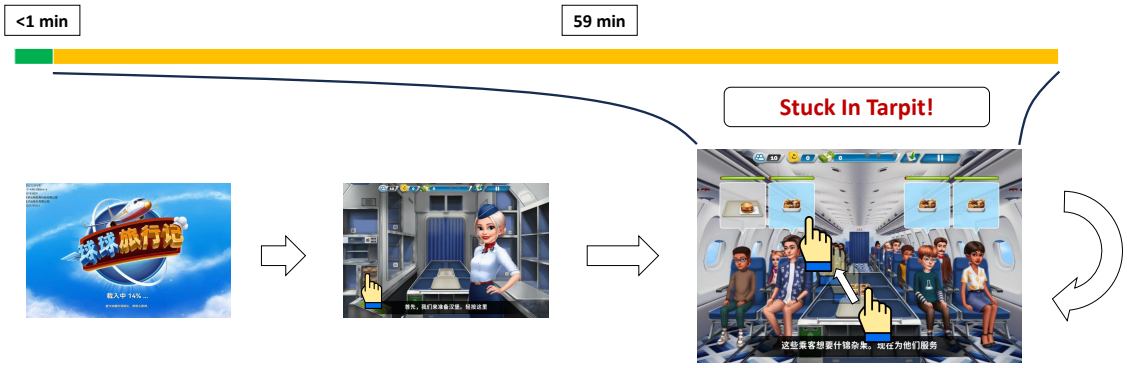

Fig. 1. A sliding-gesture-induced exploration when testing Earth Journey, a highly popular mini-game on the WeChat mini-game platform.

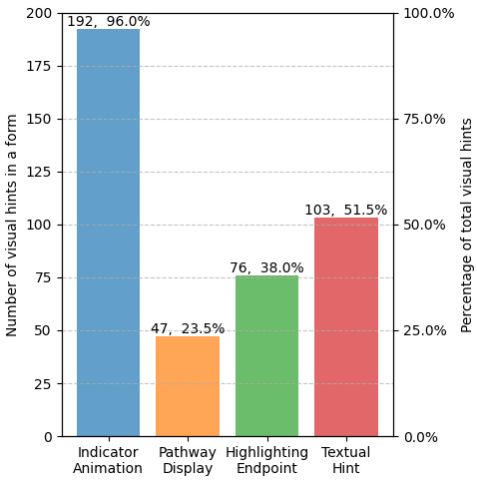

Fig. 2. Distributions of different types of visual hints in SLIDEQUIZ.

universal presence of visual hints motivates our approach to leverage these developer-provided cues for automated sliding-gesture inference. Second, indicator animation emerges as the predominant visual hint type, appearing in 192 out of 200 tarpits (96.0%). This high prevalence indicates that animated indicators are the most critical visual signal for communicating required sliding gestures, making them a primary focus for our approach.

IV. DESIGN OF SLIDESCOUT

In this section, we present the design of SLIDESCOUT, a three-stage approach for detecting and escaping sliding-gesture-induced exploration tarpits via robust and efficient exploitation of noisy indicator animation to guide proper sliding gestures for automated mini-game testing.

A. Industrial Requirements and Challenges for SLIDESCOUT

The design of SLIDESCOUT is driven by two major industrial requirements for large-scale automated mini-game testing, each presenting distinct technical challenges.

Robustness Requirement. Industrial testing environments typically provide limited, non-continuous screenshots rather than real-time video feeds, necessitating robust exploitation of visual hints from only several discontinuous screenshots. This requirement introduces three major challenges: (1) *cross-game visual diversity* - sliding indicators vary dramatically across games in appearance (arrows, glowing paths and moving objects), animation styles (linear motion, curved trajectories and pulsing effects), and subtlety (prominent vs. barely visible hints), demanding robust detection that generalizes across this heterogeneity; (2) *incomplete animation capture* - with only sparse screenshots, indicator animations may be partially captured or even missed entirely, requiring inference techniques that can reconstruct sliding trajectories from fragmentary visual evidence; and (3) *precise gesture translation* - the tool must accurately translate detected visual hints into precise sliding coordinates with appropriate parameters (start/end points, gesture speed and pressure) that satisfy each game's specific physics and interaction requirements.

Efficiency Requirement. The tool must support thousands of concurrent testing services with minimal overhead and costs, processing visual hints and generating sliding gestures rapidly enough to avoid becoming a bottleneck in high-throughput testing pipelines. This requirement presents two key challenges: (1) *real-time processing constraints* - each screenshot analysis and sliding gesture inference must complete within strict time limits (typically under 300ms) to maintain testing pipeline throughput; and (2) *resource optimization* - the approach must maintain low computational and memory footprints to enable cost-effective deployment across large-scale testing infrastructures without requiring expensive hardware acceleration.

B. Overview of SLIDESCOUT

To meet the preceding industrial requirements, SLIDESCOUT enhances testing tool effectiveness through a three-stage approach, as illustrated in Figure 3. Given a mini-game under test and an existing testing tool as inputs, SLIDESCOUT operates as follows:

Tarpit monitoring and indicator detection. In the first

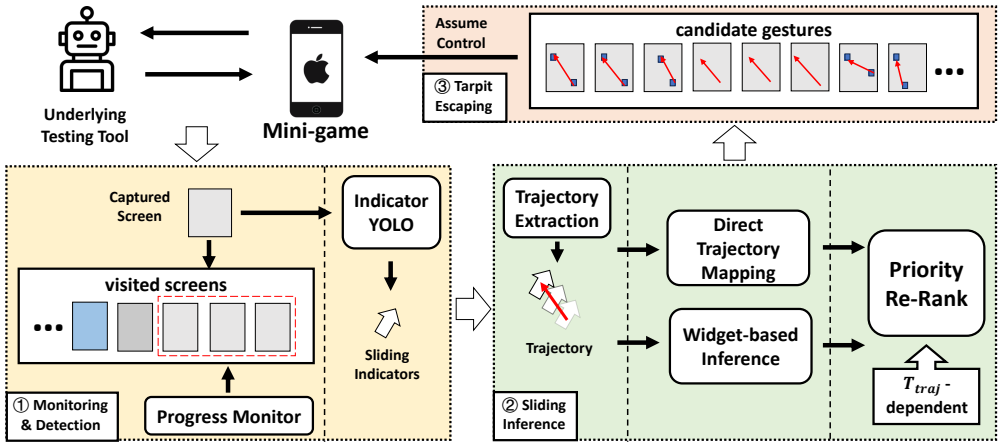

Fig. 3. Overview of SLIDESCOUT and its workflow to help any given testing tool to escape from sliding-gesture-induced exploration tarpits.

stage, SLIDESCOUT continuously monitors testing progress by collecting screenshots and analyzing state transitions. When the testing tool cycles through a limited set of game states without discovering new ones over a configurable time window, SLIDESCOUT confirms the occurrence of an exploration tarpit and triggers intervention.

Sliding-gesture inference. Upon tarpit detection, SLIDESCOUT analyzes the accumulated screenshot history to identify sliding indicators that suggest appropriate escape gestures. SLIDESCOUT employs a hybrid computer vision pipeline that combines complementary detection strategies to reliably infer sliding gestures from limited and noisy visual information, addressing the challenge of processing discontinuous screenshots typical in industrial environments.

Tarpit-escape execution. Finally, SLIDESCOUT temporarily assumes control from the underlying testing tool to execute the inferred sliding gestures. After each gesture execution, the system monitors subsequent state transitions to detect successful tarpit escape, which is indicated by the discovery of previously unexplored game states. Upon confirming successful escape, SLIDESCOUT returns control to the underlying testing tool, which resumes normal exploration with expanded state coverage.

C. Tarpit Monitoring and Indicator Detection

Tarpit monitoring. Detecting exploration tarpits requires identifying when testing tools become trapped in repetitive interaction patterns without making meaningful progress in state-space exploration. However, exploration-tarpit detection in mini-games faces a unique challenge: game screens typically incorporate animated elements and dynamic visual effects to enhance user experience, causing two screenshots from functionally identical game states to differ significantly at the pixel level.

To address this challenge, SLIDESCOUT implements a histogram-based technique for state-equivalence detection that reliably identifies functional-state repetition despite superficial visual variations. When processing each new screenshot, SLIDESCOUT calculates its color histogram and compares it against histograms of previously captured screens using the Bhattacharyya Coefficient [16]. SLIDESCOUT classifies two screenshots as representing the same functional state when their Bhattacharyya Coefficient exceeds a threshold $T = 0.5$ and declares an exploration tarpit when no new states have been discovered after $m = 10$ consecutive interaction attempts. The threshold setting strikes a balance between sensitivity (ensuring that genuine state repetitions are detected in a timely manner) and specificity (avoiding false positives caused by legitimate gameplay transitions that involve visually similar elements). Upon tarpit detection, SLIDESCOUT temporarily assumes control from the underlying testing tool to execute its specialized sliding-gesture inference (Section IV-D) and escape mechanisms (Section IV-E).

Sliding indicator detection. SLIDESCOUT identifies sliding indicators by analyzing screen captures. SLIDESCOUT employs a fine-tuned YOLO-v7 [17] network to detect indicators efficiently when a new screen capture comes.

D. Sliding-Gesture Inference

After detecting an exploration tarpit, SLIDESCOUT must infer the appropriate sliding gesture required to escape the tarpit. To overcome the intertwined challenges mentioned above, we design a multi-stage inference pipeline that first extracts meaningful trajectory information from screen captures and then proposes optimized sliding gestures through complementary strategies.

1) Extracting Trajectory from Sliding Indicators

Given limited screen captures and presented with the *cross-game visual diversity* challenge, SLIDESCOUT follows a tracking-by-detection paradigm [18] and extracts the trajectory of sliding indicators detected during the first stage by approximating the indicators by a straight line. However, even with precise detection results for each frame, the detected indicators often poorly fit into a straight-line trajectory due to incomplete

screen captures and imprecise visual guidance. Therefore, we apply a Kalman Filter [19] to the detected sliding indicators to obtain robust and stable trajectory information.

2) Proposing Candidate Sliding Gestures

Based on the extracted trajectory of sliding indicators, SLIDESCOUT proposes candidate sliding gestures using two complementary strategies.

Direct trajectory mapping: When sliding indicators provide clear and complete guidance, we map the trajectory directly to a sliding gesture. Through consultation with mini-game designers, we learn that as a design principle for better user experience, sliding indicators seldom overlap with the required sliding gestures but instead intersect with them from below (typically through an arrow or finger icon positioned at the topmost area of the indicator). Rather than simplistically using the geometric centers of sliding indicators, we utilize the Segment Anything Model (SAM) [20] to obtain segmentation masks [21] of sliding indicators and select the topmost area of the segmentation mask as the "active point", which helps mitigate the *precise gesture translation* challenge. Beyond following the exact trajectory of active points to perform sliding gestures, SLIDESCOUT also extends and adds perturbations to the trajectory as alternative candidates. These supplementary candidates provide useful gestures in situations where only a sub-segment of the complete trajectory can be inferred from limited information.

Widget-based inference: When direct trajectory mapping is insufficient due to incomplete or ambiguous trajectories as described in the *incomplete animation capture* challenge, we deploy a widget-based inference strategy. This strategy assumes that the sliding gesture connects two interactive widgets (detected by and reused from the underlying testing tool) while maintaining directional properties similar to the observed indicator. To determine the most probable sliding gesture from numerous candidates, we rank them using a comprehensive scoring function $f(s, e)$ for a candidate sliding gesture starting from s and ending at e:

$$C' = -(A \cdot x_{\mathbf{s}} + B \cdot y_{\mathbf{s}}) \tag{1}$$

$$L' : Ax + By + C' = 0 \tag{2}$$

$$f(\mathbf{s}, \mathbf{e}) = -dist(\mathbf{S}, \mathbf{s}) - dist(\mathbf{E}, \mathbf{e}) - \lambda \cdot dist(\mathbf{e}, L') \tag{3}$$

where the sliding indicator's trajectory runs from $\mathbf{S}$ to $\mathbf{E}$ with equation $Ax + By + C = 0$. By emphasizing trajectory slope over absolute position ($\lambda = 10$), our algorithm effectively ranks candidate gestures even when the estimated trace provides only approximate and incomplete trajectory guidance. As long as we obtain the correct direction from the sliding indicator, the trajectory slope factor can narrow down candidates to a manageable set.

3) Re-ranking of Candidate Sliding Gestures

In the final stage, we dynamically re-rank the candidate sliding gestures from both inference strategies to optimize success probability. Our analysis reveals complementary strengths: when sliding indicator trajectories are short, screen captures provide sufficient information for complete trajectory extraction, making direct trajectory mapping highly effective. Conversely, when trajectories are long, screen captures provide limited information (such as missing starting or ending positions), making the extracted trajectory incomplete and requiring widget-based inference with emphasis on trajectory slope similarity.

To leverage these complementary capabilities, SLIDESCOUT distinguishes between different sliding gesture classes by comparing the inferred trajectory's length against a threshold T_{traj}. When trajectory length falls below T_{traj}, we prioritize direct trajectory mapping candidates; otherwise, we prioritize widget-based inference candidates. This adaptive prioritization strategy enhances exploration efficiency by tailoring the approach to each tarpit scenario's specific characteristics.

E. Tarpit-Escape Execution

After inferring the desired sliding gestures, SLIDESCOUT executes candidate sliding gestures to escape the detected exploration tarpit. SLIDESCOUT temporarily assumes control from the mini-game testing tool and implements a systematic execution strategy for the inferred sliding gestures. This process follows a prioritized approach, beginning with the highest-confidence sliding gesture candidates identified during the inference phase. For each candidate, SLIDESCOUT translates the abstract sliding trajectory into concrete touch events with precise coordinates that simulate authentic user sliding gestures. After each sliding gesture execution, SLIDESCOUT captures the resulting screen state and analyzes it to determine whether the tarpit has been successfully navigated. Upon successful escape (indicated by the discovery of a previously unobserved state), SLIDESCOUT returns control to the underlying mini-game testing tool, allowing it to resume normal exploration from the newly discovered state. The handoff mechanism ensures that SLIDESCOUT integrates non-intrusively with any existing testing tool, precisely providing specialized sliding-gesture capabilities when needed without disrupting the overall testing workflow.

V. EVALUATION

To evaluate the effectiveness of SLIDESCOUT, we conduct extensive experiments to answer the following research questions (RQs):

- **RQ1:** How effective is SLIDESCOUT at inferring sliding gestures to escape exploration tarpits in SLIDEQUIZ?
- **RQ2:** How effective is SLIDESCOUT at improving the testing coverage of automated mini-game testing?

A. Evaluation Setup

1) Datasets.

We use SLIDEQUIZ described in Section III for evaluating the accuracy of sliding-gesture inference, enabling quantitative comparison between SLIDESCOUT and baseline approaches.

2) Baselines.

Given the strong generalization and reasoning capabilities of these large-pretrained models [22], [23], we compare SLIDESCOUT against both random approaches and VLMs in the experiments on SLIDEQUIZ.

Random (pixel-level) simply slides from a random position to another random position on the screen. We calculate metrics here as the approximate expectation over an even distribution of all possible positions.

Random (widget-level) randomly tries sliding gestures connecting two detected widgets on the screen. To reduce randomness, we randomly sample at most 100 candidates and calculate all metrics as the expectation.

VLM baselines (Qwen-VL-max [24], Qwen-VL-plus [24], GPT-4o [25]) adopt the most capable VLMs in GUI understanding. Provided with a prompt containing a sequence of screen captures, the VLMs output the top-10 most probable sliding gestures' starting and ending points in relative coordinates.

3) Evaluation Metrics.

We use two sets of evaluation metrics to measure the effectiveness of SLIDESCOUT in inferring sliding gestures and improving the performance of mini-game testing, respectively. We adopt Recall@K (R@K) and Mean Reciprocal Rank@K (MRR@K) for offline experiments in RQ1, and the Number of Aggregated Pages (NAP) for end-to-end experiments in RQ2.

Recall@K (R@K) has been widely adopted in retrieval systems. The Recall@$K = \frac{\text{\# of cases succeeding in } K \text{ attempts}}{\text{\# of all cases}}$ represents the ratio of cases where the approach can find the target sliding gesture in the first K attempts. A higher R@K indicates that the approach infers more effective candidate gestures.

Mean Reciprocal Rank@K (MRR@K) focuses on the highest-rank positive samples. It is calculated as the average of the reciprocal rank of the first effective candidate sliding gesture, as follows:

$$\text{MRR@}K = \frac{1}{|\mathbf{C}|}\Sigma_{c\in\mathbf{C}}\frac{1}{rank_c}$$

where $\mathbf{C}$ represents the set of test cases and $rank_c$ represents the rank of the first effective candidate sliding gesture. If no candidate succeeds within K attempts, $\frac{1}{\text{rank}_c}$ is set to 0. Higher MRR@K reflects better prioritization of correct gestures.

Number of Aggregated Pages (NAP) quantifies the exploration capability of a testing tool by counting unique screen pages encountered. It has been widely used in the WeChat mini-game testing platform [7] to measure the exploration ability of a tool. Only pages whose similarity score (which is calculated by comparing the histogram with Bhattacharyya coefficient) with all prior pages falls below a threshold are considered as a new non-repeating page and contribute to the NAP metric.

4) Underlying Mini-game Testing Tool.

iExplorerGame [7], the mini-game testing tool that WeChat platform has deployed, identifies and interacts with mini-game widgets through deep-learning-based object detection and interaction-type categorization. While iExplorerGame generates sliding gestures mostly at random, it fails to perform specific sliding gestures indicated by visual hints and frequently gets stuck in exploration tarpits during deployment at WeChat. We integrate SLIDESCOUT into this tool and enhance its tarpit-escaping ability.

5) Implementation Details.

Given the industrial efficiency requirement, we aim to carefully choose small-sized but effective models. We train our sliding indicator detector by fine-tuning YOLO-v7 [17] on a set of 2,000 pictures with manual annotations on an NVIDIA T4 with 16GB of graphics memory, using the same parameter settings as the original repository. In the direct trajectory mapping strategy, we choose SAM2-tiny [20], which not only performs well enough for our task but also runs efficiently. Both models have fewer than 40M parameters while maintaining the capability to handle corresponding tasks.

B. RQ1: Effectiveness of SLIDESCOUT in Inferring Sliding Gestures

1) Main Results.

To evaluate the effectiveness of SLIDESCOUT in inferring sliding gestures, we conduct experiments on our SLIDEQUIZ dataset. The results of accuracy in inferring sliding gestures are presented in Table I. From the table, we have three major observations.

First, SLIDESCOUT substantially outperforms all the baseline approaches across all evaluation metrics. Our approach achieves a 37.0% success rate for its top-ranked prediction (R@1), which is over 8 times higher than the best baseline approach. This result demonstrates that our specialized sliding-gesture inference pipeline effectively captures and interprets sliding indicators in mini-games. The performance advantage becomes even more pronounced when considering multiple attempts, with SLIDESCOUT reaching a 64.5% success rate within ten attempts (R@10), compared to only 8.5% for the best baseline.

Second, the results highlight the difficulty for general-purpose VLMs to perform real-world tasks alone without domain knowledge [26], [27]. Advanced VLMs like Qwen-VL-max, despite effective in general visual understanding tasks, achieve only 3% accuracy for top predictions on this specialized task. It is worth noting that the other two VLMs, GPT-4o [25] and Qwen-VL-plus [24], both fail to predict meaningful sliding gestures in our experiments and are shown as 0.0% in the results. This poor performance underscores the unique challenges in interpreting game-specific visual hints and translating them into precise sliding coordinates.

The strong MRR@10 score of 0.453 further confirms that correct predictions consistently appear near the top of SLIDESCOUT's ranked suggestions. These results validate our approach's effectiveness in tackling the challenging task of sliding-gesture inference from discontinuous visual hints, addressing a significant barrier in automated mini-game testing that previous approaches have largely failed to overcome.

TABLE I
PERFORMANCE COMPARISON OF SLIDING GESTURE INFERENCE APPROACHES

Approach	R@1	R@3	R@5	R@10	MRR@10
Random(pixel-level)	0.03%	0.11%	0.19%	0.38%	0.001
Random(widget-level)	4.00%	4.50%	5.00%	8.50%	0.082
VLM (Qwen-VL-max)	3.00%	3.50%	4.50%	6.00%	0.036
VLM (Qwen-VL-plus)	0.00%	0.00%	0.00%	0.00%	0.000
VLM (GPT-4o)	0.00%	0.00%	0.00%	0.00%	0.000
SLIDESCOUT	**37.00%**	**46.00%**	**59.00%**	**64.50%**	**0.453**

TABLE II
PERFORMANCE COMPARISON OF SLIDING GESTURE PROPOSAL STRATEGIES

Strategy	Recall@1	Recall@3	MRR@10
Element-based only	27.0%	37.5%	0.376
Trajectory-based only	36.0%	44.0%	0.443
SLIDESCOUT (Combined)	**37.0%**	**46.0%**	**0.453**

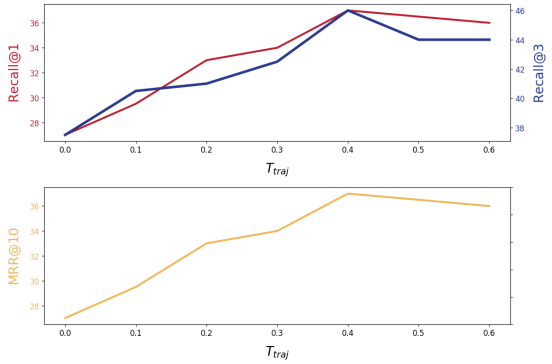

Fig. 4. Impact of trajectory confidence threshold T_{traj} on sliding-gesture inference accuracy.

2) Ablation on Complementary Strategies for Proposing Sliding Gestures.

Our analysis focuses on understanding the complementary nature of our dual proposal strategies: direct trajectory mapping and widget-based inference. To understand the complementary nature of our dual strategies, we analyze their individual and combined performance on our benchmark dataset, as presented in Table II. The results reveal strong complementarity between our two strategies as shown in Section IV-D3.

We further calibrate the trajectory confidence threshold T_{traj} across a range from 0.1 to 0.6 to understand its impact on overall performance. As shown in Figure 4, a threshold of $T_{traj} = 0.4$ yields optimal overall performance for SLIDESCOUT. This threshold balances the contributions of both proposal strategies to maximize accuracy across diverse sliding-gesture challenges, indicating that our parameter setting of T_{traj} represents a balanced trade-off.

3) Failure Analysis.

Despite the substantial improvement over the baseline approaches, there are still tasks that SLIDESCOUT fails to efficiently infer. In 71 cases where SLIDESCOUT fails to perform the required sliding gesture in the first 10 attempts, the required gesture of 18 of them can be found by SLIDESCOUT when given more chances (found in 50 attempts). For the remaining 53 cases, we conduct a detailed analysis of failure cases below, identifying four primary categories of challenges.

Non-linear sliding gesture trajectories (13.2% of failures): Our current trajectory extraction algorithm assumes predominantly linear sliding gesture trajectories, being inadequate for complex curved or multi-segment sliding trajectories. In games requiring arc-shaped swipes, zigzag patterns, or circular motions, SLIDESCOUT often fails to trigger the intended interaction.

Temporally complex sliding indicators (41.5% of failures): Some games present sliding indicators through elaborate multi-phase animations that require temporal understanding beyond what our current system can extract from discontinuous screenshots. These animations include sequences where the indicator exhibits pulsing behavior or appears in different shapes and sizes at different times. Limited screenshots often capture only fragments of these complex temporal patterns, leading to incomplete trajectory inference.

Camouflaged or subtle sliding indicators (24.5% of failures): Certain games deliberately employ subtle or environmentally integrated sliding indicators that blend with background elements. Our current sliding indicator detection focuses primarily on high-contrast, animated elements and frequently misses these camouflaged indicators, resulting in completely missed trajectories.

Compound visual hints for sliding gestures (20.8% of failures): In some cases, the correct sliding gesture depends not just on the sliding indicator but on other forms of visual hints or a mixture of different hint types. SLIDESCOUT currently lacks the ability to incorporate such compound visual hints, leading to incorrect trajectory inferences in state-dependent scenarios.

The failure analysis highlights important directions for future work. Addressing non-linear trajectories through more sophisticated curve-fitting algorithms, improving temporal pattern recognition across discontinuous captures, enhancing detection of subtle sliding indicators, and incorporating compound visual hint understanding would significantly expand SLIDESCOUT's coverage of challenging sliding-gesture scenarios.

C. RQ2: Improvement of End-to-End Testing Effectiveness with SLIDESCOUT

To evaluate how SLIDESCOUT's sliding-gesture inference capabilities translate to real-world testing improvements, we integrate SLIDESCOUT with the industrial-grade automated mini-game testing framework named iExplorerGame [7] and conduct comprehensive end-to-end testing experiments across diverse mini-games. For each test on a mini-game, we run the testing tool for one hour. Due to the random factors in testing, e.g., screens captured and mini-game runtime randomness, we test the underlying tool with or without SLIDESCOUT for three times on each mini-game and calculate an average NAP.

TABLE III
PERFORMANCE IMPROVEMENT BY SLIDESCOUT ON BASELINE MINI-GAME TESTING PIPELINE

Mini-Game	w\o SLIDESCOUT	with SLIDESCOUT	
	Average NAP	**Average NAP**	**Tarpit Pass Rate**
Average	21.61	**30.21** (+39.80%)	**40.48%**

1) Main Results.

Table III presents the end-to-end testing results. From the table, we have three major observations. First, SLIDESCOUT dramatically improves testing effectiveness. Compared to original iExplorerGame without SLIDESCOUT, SLIDESCOUT increases screen coverage by 39.80% (from 21.61 to 30.21 screens per game), a substantial improvement over the underlying testing tool that lacks specialized sliding-gesture handling mechanisms.

Second, a detailed analysis of the testing sessions reveals that SLIDESCOUT's improvements are particularly pronounced in games with sliding challenges in the early stage. In these scenarios, successfully navigating one sliding tarpit often unlocks access to abundant additional game content. For example, in a tower defense game requiring five sequential sliding gestures to progress, the underlying testing tool explores only the first game level (9 screens), while SLIDESCOUT helps the testing tool successfully navigate through 3 to 4 tarpits and explore 43 unique screens.

Third, the 40.48% tarpit escape rate, despite a substantial improvement, indicates room for further enhancement. Our analysis of the remaining unresolved tarpits aligns with the failure categories identified in RQ1, with non-linear gestures and complex temporal hints being particularly challenging in end-to-end testing scenarios. Additionally, some sliding tarpits occur in sequence where exploring more game content requires to pass more than one tarpit, demanding higher precision of sliding gesture inference to escape the tarpit.

These results demonstrate that SLIDESCOUT substantially improves automated testing effectiveness for mini-games containing sliding-gesture tarpits, enabling exploration of previously inaccessible game content and significantly expanding test coverage.

2) Efficiency of SLIDESCOUT

Our analysis reveals that SLIDESCOUT introduces negligible computational overhead to the testing process. From obtaining the sequence of captures to generating sliding gesture candidates, the inference component requires less than 0.1 seconds on average to process visual hints and prioritize trajectory predictions. The deep learning models used in SLIDESCOUT, YOLO-v7 [17] and SAM2-tiny [20], both have less than 40M parameters and meet the platform's efficiency requirement with redundancy. This ultra-efficient inference time is achieved through our three optimizations in SLIDESCOUT: (1) choosing a lightweight indicator detection pipeline, (2) applying a small-sized but capable segmentation and detection model, and (3) concurrently detecting indicators when monitoring the testing process. When the underlying testing tool is stuck in tarpit, compared to the wasted time that the underlying tool spends repeating meaningless gestures, SLIDESCOUT's 0.1-second inference overhead per tarpit is virtually a no-cost solution. Moreover, thanks to our monitoring system, the main process of SLIDESCOUT is called only when the underlying testing tool becomes truly stuck and produces almost zero overhead at the scale of the entire testing process.

VI. DEPLOYMENT AND LESSONS LEARNED

A. Deployment of SLIDESCOUT

Since January 2025, we have deployed SLIDESCOUT at the testing platform for WeChat, one of the most popular mobile apps with over one billion monthly active users. WeChat operates as one of the largest mini-game platforms, hosting over 100,000 mini-games, with thousands of new mini-games undergoing automated testing daily on the platform before public release. SLIDESCOUT has been integrated as the default strategy for handling exploration tarpits across all mini-game testing workflows on the platform. During the six-month deployment period, SLIDESCOUT has been invoked over 500,000 times, demonstrating its scalability and reliability in high-throughput industrial environments. The deployment has yielded significant improvements in testing effectiveness, helping reveal more than 25,000 game crashes and 120,000 JavaScript errors that would have otherwise remained undetected due to exploration tarpits preventing thorough game state coverage. To evaluate the practical impact of SLIDESCOUT, we compare testing outcomes against pre-deployment baselines from the same six-month period in the previous year. The results show that SLIDESCOUT improves the effectiveness of crash detection by 50% and JavaScript error detection by 17%, respectively. These improvements directly translate to enhanced game quality and reduced post-release issues, validating the industrial value of SLIDESCOUT for large-scale automated mini-game testing.

B. Lessons Learned

We summarize three major lessons learned from developing and deploying SLIDESCOUT.

1) We should match the right solution to the right scale.

The unprecedented scale of testing mini-games at WeChat fundamentally reshapes both the solution design space and optimization goals.

Runtime costs matter at scale. With thousands of devices conducting round-the-clock testing across 3,000+ daily tasks, efficiency optimizations that seem negligible for single-application testing become critical for system viability. This factor drives multiple design decisions in SLIDESCOUT. First, we instantiate SLIDESCOUT with efficient models of object detection (e.g., TinySAM [20] and YOLO-v7 [17]) rather than computationally expensive large models [25]. Second, we concurrently process and cache detection results across pipeline stages, achieving near-zero additional overhead for the testing process.

Edge-case handling matters at scale. In single-application testing, occasional failures can be addressed through manual intervention or application-specific patches. However, at WeChat scale with 3,000+ daily testing tasks, even a 1% failure rate becomes unmanageable. This factor drives the need for SLIDESCOUT to handle previously niche scenarios that become statistically significant at scale.

Generalization becomes a first-class citizen. Single-application testing can rely on application-specific heuristics and domain knowledge, but platform-scale testing demands solutions that generalize across thousands of distinct applications. For example, sliding indicators and guidance vary dramatically across games. To improve the generalization, SLIDESCOUT performs robust trajectory reconstruction by combining multiple and complementary inference strategies.

2) We must align solutions with infrastructure constraints.

While continuous screen captures would dramatically reduce the difficulty of sliding-gesture-induced tarpit handling, the required hardware infrastructure (high-speed cameras and data transmission for thousands of devices) is economically prohibitive at WeChat scale. Rather than pursuing ideal but unaffordable solutions, we learn to work within existing infrastructure and develop robust inference algorithms for discontinuous screenshots, demonstrating how industrial scale transforms theoretical advantages into practical impossibilities.

3) There is a need to investigate individual cases beyond benchmark metrics.

Rather than evaluating different approaches solely on benchmark metrics, there is a need to investigate individual cases for deep insight.

General-purpose VLMs can fall short in domain-specific problems. While general-purpose VLMs [28], [29], [30], [31] have shown great potential in many general GUI testing benchmarks [32], [33], [34], our deployment and evaluation results demonstrate that state-of-the-art VLMs [24], [25] fall far short of our expectations in mini-game testing, where VLMs' performance is on a par with simple heuristics, as presented in Table I. With a deeper investigation into their failure cases, we discover that while the responses of VLMs indicate that they can understand the task and game-screen captures in a high-level sense, VLMs frequently fail to generate meaningful sliding gestures but abstract gameplay instructions or unreasonable random horizontal sliding gestures without grounding to the game UIs. This detailed analysis reveals critical limitations that would be missed by benchmark metrics alone.

Failure-case analysis provides significant insight. It is important to investigate failure cases of existing approaches to identify root issues and further enhance capabilities. In practice, we examine mini-game cases where the existing testing tool achieves limited coverage and collect their testing logs directly from the WeChat mini-game testing platform. Through an in-depth analysis of these trajectories, we find that the low efficiency of the preceding testing tool is not simply due to the tool's inherent limitations, but rather the existence of exploration tarpits (in mini-game testing) that exceed the tool's capabilities. The experience demonstrates that comprehensive failure analysis on individual cases is essential to identify underlying defects and mitigate them, particularly in industrial settings where practical circumstances frequently deviate from theoretical expectations, as noted in other studies [8], [35].

VII. RELATED WORK

Exploration tarpits in automated UI testing. Vet [12] first formalizes the concept of exploration tarpits for mobile apps, identifying scenarios where testing tools repeatedly explore the same states without making meaningful progress. To mitigate the impact of exploration tarpits, Vet applies trace analysis to identify the UI elements for exploration tarpits and disables the UI elements that might lead to tarpits. Aurora [14] applies deep learning to recognize and bypass tarpits. Our work extends these concepts by specifically targeting the unique challenges of mini-games, where we leverage visual hints to guide automated testing through complex sliding gestures that conventional approaches do not tackle.

Automated UI testing for mobile apps and games. While automated UI testing for mobile apps has been extensively studied [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [26], [48] and widely adopted in industry [49], [8], [50], automated testing for games remains comparatively underexplored. Existing game testing approaches typically produce only preliminary findings [51], focus on detecting only a certain kind of bug [52], or rely on developer-provided information such as game states [53], [54] and internal APIs [55]. The few generic approaches [56], [7] focus primarily on completing basic testing workflows through computer-vision-based element detection and interaction, but struggle with specialized interactions unique to games.

Our approach addresses a critical gap by focusing specifically on sliding gestures, a common yet challenging interaction pattern in mini-games that existing work cannot effectively tackle. The tool-agnostic strategy of SLIDESCOUT makes it general to enhance any testing tool for improving its effectiveness in the diverse mini-game ecosystem.

VIII. CONCLUSION

In this paper, we have reported our experiences and lessons learned from developing and deploying SLIDESCOUT for automated detection and escape of sliding-gesture-induced exploration tarpits in mini-game testing. SLIDESCOUT exploits visual hints, particularly indicator animations, present in game UIs to systematically infer precise sliding gestures, meeting the robustness and efficiency requirements for industrial-scale mini-game testing. Deployed at WeChat for six months, SLIDESCOUT helps reveal 25,000 crashes and 120,000 JavaScript errors, detecting 50% more crashes compared to pre-deployment baselines within the same time period. We have summarized three major lessons learned from developing and deploying SLIDESCOUT.

ACKNOWLEDGMENTS

Tao Xie is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing, China; Fudan University Institute of Systems for Advanced Computing, Shanghai, China; Shanghai Institute of Systems for Open Computing, Shanghai, China. Dezhi Ran and Tao Xie are partially supported by the National Natural Science Foundation of China under Grant Nos. 623B2006 and 92464301. Dezhi Ran is partially supported by a Hunyuan Scholar Award.

REFERENCES

[1] WeChat Wiki, "WeChat mini-game development," https://wechatwiki.com/wechat-resources/wechat-mini-game-development/, 2019.

[2] M. Steinberg, R. Mukherjee, and A. Punathambekar, "Media power in digital Asia: Super apps and megacorps," *Media, Culture & Society*, vol. 44, no. 8, 2022.

[3] Y. Zhang, B. Turkistani, A. Y. Yang, C. Zuo, and Z. Lin, "A measurement study of WeChat mini-apps," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 2, 2021.

[4] T. SocialPeta, "SocialPeta x Tenjin — insights into global mobile game marketing & Ad spend trends for H1 2024," https://socialpeta.com/academy/socialpeta-tenjin-insights-into-global-mobile-game-marketing-ad-spend-trends-for-h1-2024, 2024.

[5] 36Kr English, "China's minigame boom: A fading opportunity for most," https://kr-asia.com/chinas-minigame-boom-a-fading-opportunity-for-most, 2024.

[6] China audio-video and digital publishing association, "2024 China game industry report," http://www.cadpa.org.cn/3277/202501/41718.html, 2024.

[7] C. Wang, H. Lu, C. Gao, Z. Li, T. Xiong, and Y. Deng, "A unified framework for mini-game testing: Experience on WeChat," in *FSE*, 2023.

[8] D. Ran, Z. Li, C. Liu, W. Wang, W. Meng, X. Wu, H. Jin, J. Cui, X. Tang, and T. Xie, "Automated visual testing for mobile apps in an industrial setting," in *ICSE-SEIP*, 2022.

[9] Z. Song, Y. Chen, L. Ma, S. Lu, H. Lin, C. Fan, and W. Yang, "An empirical analysis of compatibility issues for industrial mobile games (practical experience report)," in *ISSRE*, 2022.

[10] H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan, "Prioritizing the devices to test your app on: A case study of Android game apps," in *FSE*, 2014.

[11] J. Tuovenen, M. Oussalah, and P. Kostakos, "MAuto: Automatic mobile game testing tool using image-matching based approach," *The Computer Games Journal*, vol. 8, 2019.

[12] W. Wang, W. Yang, T. Xu, and T. Xie, "Vet: identifying and avoiding UI exploration tarpits," in *FSE*, 2021.

[13] G. Lovreto, A. Endo, P. Nardi, and V. Durelli, "Automated tests for mobile games: An experience report," in *SBGames*, 2018.

[14] S. A. Khan, W. Wang, Y. Ren, B. Zhu, J. Shi, A. McGowan, W. Lam, and K. Moran, "Aurora: Navigating UI tarpits via automated neural screen understanding," in *ICST*, 2024.

[15] ASE-RISE, "Sliderscout tool," https://github.com/PKU-ASE-RISE/SliderScout, 2025.

[16] A. Bhattacharyya, "On a measure of divergence between two multinomial populations," *Sankhyā: The Indian Journal of Statistics (1933-1960)*, vol. 7, no. 4, 1946.

[17] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022. [Online]. Available: https://arxiv.org/abs/2207.02696

[18] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," in *ICCV*, 2008.

[19] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[20] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "SAM 2: Segment anything in images and videos," 2024. [Online]. Available: https://arxiv.org/abs/2408.00714

[21] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," 2020. [Online]. Available: https://arxiv.org/abs/2001.05566

[22] D. Ran, M. Wu, W. Yang, and T. Xie, "Foundation model engineering: Engineering foundation models just as engineering software," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, 2025.

[23] D. Ran, Y. Cao, Y. Guo, Y. Li, M. Wu, S. Chen, W. Yang, and T. Xie, "Medusa: A framework for collaborative development of foundation models with automated parameter ownership assignment," in *FSE*, 2025.

[24] J. Bai, S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou, and J. Zhou, "Qwen-VL: A versatile vision-language model for understanding, localization, text reading, and beyond," 2023. [Online]. Available: https://arxiv.org/abs/2308.12966

[25] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida *et al.*, "GPT-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[26] D. Ran, H. Wang, Z. Song, M. Wu, Y. Cao, Y. Zhang, W. Yang, and T. Xie, "Guardian: A runtime framework for LLM-based UI exploration," in *ISSTA*, 2024.

[27] D. Ran, M. Wu, Y. Cao, A. Marron, D. Harel, and T. Xie, "An infrastructure software perspective toward computation offloading between executable specifications and foundation models," *Science China Information Sciences*, vol. 68, no. 4, 2025.

[28] S. Wang, S. Wang, Y. Fan, X. Li, and Y. Liu, "Leveraging large vision-language model for better automatic web GUI testing," in *ICSME*, 2024.

[29] W. Hong, W. Wang, Q. Lv, J. Xu, W. Yu, J. Ji, Y. Wang, Z. Wang, Y. Dong, M. Ding, and J. Tang, "CogAgent: A visual language model for GUI agents," in *CVPR*, 2024.

[30] X. H. Lù, Z. Kasner, and S. Reddy, "WEBLINX: Real-world website navigation with multi-turn dialogue," in *ICML*, 2024.

[31] K. Cheng, Q. Sun, Y. Chu, F. Xu, L. YanTao, J. Zhang, and Z. Wu, "SeeClick: Harnessing GUI grounding for advanced visual GUI agents," in *ACL*, 2024.

[32] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2Web: Towards a generalist agent for the web," in *NIPS*, 2023.

[33] C. Rawles, A. Li, D. Rodriguez, O. Riva, and T. Lillicrap, "Android in the wild: A large-scale dataset for Android device control," in *NIPS*, 2023.

[34] T. T. Shi, A. Karpathy, L. J. Fan, J. Hernandez, and P. Liang, "World of bits: An open-domain platform for web-based agents," in *ICML*, 2017.

[35] D. Ran, L. Li, L. Zhu, Y. Cao, L. Zhao, X. Tan, G. Liang, Q. Wang, and T. Xie, "Efficient and robust security-patch localization for disclosed oss vulnerabilities with fine-tuned LLMs in an industrial setting," in *FSE Companion*, 2025.

[36] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for Android apps," in *FSE*, 2013.

[37] R. Mahmood, N. Mirzaei, and S. Malek, "EvoDroid: Segmented evolutionary testing of Android apps," in *FSE*, 2014.

[38] A. Romdhana, A. Merlo, M. Ceccato, and P. Tonella, "Deep reinforcement learning for black-box testing of Android apps," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, 2022.

[39] Google, "Android Monkey," https://developer.android.com/studio/test/other-testing-tools/monkey, 2021.

[40] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps," in *MobiSys*, 2014.

[41] M. Pan, A. Huang, G. Wang, T. Zhang, and X. Li, "Reinforcement learning based curiosity-driven testing of Android applications," in *ISSTA*, 2020.

[42] S. Yu, C. Fang, Y. Yun, and Y. Feng, "Layout and image recognition driving cross-platform automated mobile testing," in *ICSE*, 2021.

[43] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, "Guided, stochastic model-based GUI testing of Android apps," in *FSE*, 2017.

[44] T. D. White, G. Fraser, and G. J. Brown, "Improving random GUI testing with image-based widget detection," in *ISSTA*, 2019.

[45] J. Eskonen, J. Kahles, and J. Reijonen, "Automating GUI testing with image-based deep reinforcement learning," in *ACSOS*, 2020.

[46] D. Ran, H. Wang, W. Wang, and T. Xie, "Badge: Prioritizing UI events with hierarchical multi-armed bandits for automated UI testing," in *ICSE*, 2023.

[47] D. Ran, Y. Fu, Y. He, T. Chen, X. Tang, and T. Xie, "Path toward elderly friendly mobile apps," *Computer*, vol. 57, no. 6, 2024.

[48] D. Ran, Z. Song, W. Wang, W. Yang, and T. Xie, "TaOPT: Tool-agnostic optimization of parallelized automated mobile UI testing," in *ASPLOS*, 2025.
[49] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in *ISSTA*, 2016.
[50] H. Zheng, D. Li, B. Liang, X. Zeng, W. Zheng, Y. Deng, W. Lam, W. Yang, and T. Xie, "Automated test input generation for Android: towards getting there in an industrial case," in *ICSE-SEIP*, 2017.
[51] J. Ye, K. Chen, X. Xie, L. Ma, R. Huang, Y. Chen, Y. Xue, and J. Zhao, "An empirical study of GUI widget detection for industrial mobile games," in *FSE*, 2021.
[52] K. Chen, Y. Li, Y. Chen, C. Fan, Z. Hu, and W. Yang, "GLIB: Towards automated test oracle for graphically-rich applications," in *FSE*, 2021.
[53] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," in *ASE*, 2019.
[54] Y. Wu, Y. Chen, X. Xie, B. Yu, C. Fan, and L. Ma, "Regression testing of massively multiplayer online role-playing games," in *ICSME*, 2020.
[55] L. Shi, Q. Ding, J. Hou, B. Zhou, C. Jin, Y. Tao, J. Wei, and S. Li, "WemiEnv: An open-source reinforcement learning platform for WeChat mini-games," *IEEE Transactions on Games*, vol. 17, no. 3, 2025.
[56] X. Wu, J. Ye, K. Chen, X. Xie, Y. Hu, R. Huang, L. Ma, and J. Zhao, "Widget detection-based testing for industrial mobile games," in *ICSE-SEIP*, 2023.