ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 5 Issue 4 October 2025 / Page No: 128-137
Paper Id: JETA-V5I4P119 / Doi: 10.56472/25832646/JETA-V5I4P119

Original Article

FlowMind: Mining Real User Telemetry to Power LLM-Driven Autonomous App Testing

Sachin Francis
Independent.

Received Date: 17 September 2025 Revised Date: 13 October 2025 Accepted Date: 14 November 2025

Abstract: *Automated testing is the cornerstone of software reliability. But authoring and maintaining functional regression tests continues to demand significant manual effort. Traditional approaches such as Espresso and Appium require engineers to script explicit user interactions into the tests. This rapidly becomes brittle as product features evolve. At the same time, every modern application already captures extensive telemetry data. Those include, but not limited to, screen impressions, navigations and user interactions. This represents a detailed record of real user behavior on the app.*

FlowMind leverages this untapped data source to enable autonomous regression testing without manual test derivation and coding / scripting. By mining tracking and telemetry logs to identify the most frequent user flows, FlowMind generates a structured schema describing real interaction sequences. A semantic repository links telemetry identifiers to human-understandable UI components, allowing a large language model driven agent to interpret and execute these flows directly within the application. The system autonomously navigates, validates, and adapts to UI changes, achieving realistic and evolving test coverage aligned with production usage patterns. A prototype implementation demonstrates that FlowMind achieves comparable coverage to manually authored tests while reducing creation and maintenance effort by more than 80%. FlowMind points toward a new paradigm of tracking / telemetry-driven, self-evolving testing.

Keywords: *Automated Testing; Regression Testing; User Telemetry; Test Generation; Large Language Models; Software Quality Assurance; Autonomous Agents; Mobile Applications; Android Testing.*

I. INTRODUCTION

Functional regression testing is central to software reliability. As applications evolve, engineers must ensure that existing user flows on the product continue to behave as expected for each new change / feature introduced. Modern automation frameworks provide programmable ways to interact with user interfaces. But they still depend on developer written code that automates the expected sequence of screen navigation, inputs or actions, and then the assertions. This also needs engineers or product owners to come up with a golden set of test cases which are to be automated. Maintaining these test case sets and automated tests across frequent releases and product iterations becomes a significant ongoing cost and adds to the developer frustration.

But we must note that in parallel, almost every production application already emits a continuous stream of telemetry. Typical tracking pipelines record screen impressions, navigation events, button clicks, text inputs, scrolling actions, other interactions to drive analytics and product decisions. This data captures what users do inside the application at scale. It therefore contains an implicit catalogue of real user flows that are known to matter in practice. But this information is rarely used to drive test automation and often not event used in the manual derivation of test cases which must be automated.

FlowMind explores how to bridge these two worlds. It treats tracking / telemetry as a first-class test signal and uses it to derive and execute realistic regression checks without manual test scripting.

A. Background and Related Work

a) Motivation

The disconnect between how users interact with an application and how automated tests are authored is significant. Teams invest considerable effort writing and maintaining literal human understandable tests and the scripted tests that attempt to mimic expected usage patterns. These tests are often based on assumptions and initial contract or spec rather than on real behavior. As a result, regression suites do not always reflect the actual flows that matter the most to users. They cover what engineers believe to be representative use flows rather than what users perform at scale. Recent research demonstrates that large language models can autonomously explore and interact with mobile user interfaces [1], [2], [5], but these approaches are not grounded in real user telemetry and therefore do not prioritize the flows that matter most in production. While it's good to have all combinations of a new feature spec to be automated, it comes at a cost. Some of these features many

even be in the top 99% user flows and running these tests for every commit makes build process slow and ends up with wasted resources.

This situation becomes more challenging as applications grow in complexity. Mobile products evolve rapidly and introduce new features, new screens, and new navigation paths in each release. A single commit can invalidate an entire group of tests making it obsolete. The maintenance burden falls on developers and quality engineers who must repeatedly update tests, repair assertions, or rewrite entire sequences. These do not produce new product value and lag behind the releases, making product vulnerable to delayed detection of bugs.

In the parallel dimension, the telemetry pipeline continues to capture the exact flows users exercise every day. It records which screens users visit in its order, how they transition between features, where they spend time, and which interactions lead to successful or abandoned session. These tracking events reveal high value flows such as feed consumption, profile navigation, content creation, and action completion. They also expose failure points and rare sequences that automated tests often miss.

This creates a strong motivation to rethink how functional tests are derived. Instead of building synthetic flows and attempting to keep them up to date, teams can use the data already present in the tracking system. Telemetry contains precise information about actual behavior and can serve as a naturally evolving source for test generation. If this information can be extracted in a structured form and executed automatically, functional testing becomes aligned with real user behavior. And the need for manual test authoring is significantly reduced.

FlowMind is motivated by this opportunity. It seeks to use real world interaction data to guide dynamic test derivation and test execution. By grounding regression checks in actual usage, it aims to achieve more reliable and meaningful functional validation with far less manual effort and faster build pipelines.

b) Limitations of Existing Techniques

Existing approaches for functional and user interface testing fall into several categories, but none of them leverage real user telemetry as a primary source of test derivation. As a result, they suffer from serious limitations when applications evolve rapidly or when test suites must reflect actual usage patterns.

The first category consists of code-based automation frameworks such as Espresso, UI Automator, and Appium. These frameworks require engineers to write tests that explicitly describe navigation sequences, actions, and assertions in code. They add to substantial maintenance burden. Test suites degrade over time because changes in identifiers, layouts, or navigation models invalidate entire sets of scripts. Teams must continuously update test classes. This effort grows in proportion to app complexity and release velocity. It is often disconnected from the behavior users exhibit in production.

Second category includes record and replay tools that attempt to reduce manual test case coding by capturing user interactions on a device and converting them into executable test scripts. These tools are sensitive to even small changes in UI or configuration. Recorded flows quickly become obsolete and replay process remains dependent on brittle coordinate-based strategies. Because they rely on literal user demonstration rather than mined behavior, they cannot scale to represent the broad set of flows users perform in practice.

Third category consists of random or search driven exploration tools such as Monkey style event generators or search based frameworks that attempt to traverse the application space. These approaches can uncover crashes and robustness issues but do not target meaningful user flows. They provide little control over flow selection. They do not guarantee coverage of high value flows. Hence, not suitable for regression validation where correctness and determinism are required.

Finally, recent addition is the agent-based testing methods where large language models reason over user interface structure or screenshots to autonomously explore applications. These techniques are not grounded in real user behavior and do not consume a structured representation of user flows derived from telemetry data. While autonomous LLM-based UI testing has recently emerged [1], [2], [5], these techniques operate on screenshot or structure-based reasoning and do not leverage usage analytics to guide regression intent.

Across all these categories, the fundamental limitation is the absence of a direct connection between real world app usage pattern and automated testing. None of these techniques transform tracking events into reusable flow specifications and none can maintain test suites that evolve in tandem with real user behavior. This gap motivates the design of FlowMind, which uses telemetry as a first-class source of truth for deriving and executing meaningful regression tests.

c) Problem Statement

Given an application that already emits detailed tracking events for screen impressions, navigations, and interactions, current testing practice still depends on manually written test cases and automated flows that loosely reflect real usage. There

is no systematic way to transform the rich information present in telemetry into executable functional tests. As a result, regression suites are costly to maintain. They do not consistently represent the flows that matter most in production. It also slows down build pipelines by exercising low value scenarios at the same rate as high value ones.

The problem addressed in this work is how to automatically derive and maintain functional regression tests from real user telemetry in a way that is selective, meaningful, and robust. To be very specific, we seek a framework that can:

- continuously mine tracking to identify an evolving set of representative user flows that reflect the most frequent and business critical behavior
- express these flows in a structured and reusable form that can be consumed by LLMs
- execute the flows on the application under test without requiring manual derivation of tests or automated tests

FlowMind treats this as a transformation and execution challenge. The transformation step converts raw telemetry into an abstract description of user flows that evolves as usage behavior shift. The execution step uses a LLM driven agent to realize these flows on real builds of the application and to determine whether they completed successfully. The goal is to align regression testing with actual user behavior, reduce the burden of manual test authoring, and improve build efficiency by focusing validation on flows grounded in real usage rather than static assumptions or spec driven test plans.

II. MATERIALS AND METHODS

A. Proposed Approach

a) *Overview*

FlowMind is designed to bridge the gap between real user behavior and automated regression testing by transforming tracking data into executable test flows. It introduces a telemetry centered pipeline that extracts meaningful usage patterns, converts them into a structured schema, and delegates execution to a large language model driven agent without requiring any manual test scripting. At a high level, FlowMind operates in three stages.

- The first stage is the telemetry analysis layer. It continuously collects and processes tracking events that represent screen views, navigations, and user interactions. These events are aggregated to identify the most frequent and business critical user flows. The output of this stage is an evolving set of representative usage sequences that reflect how users navigate and interact with the application.
- The second stage is the transformation layer. FlowMind converts each identified flow into a structured contract. This contract describes the sequence of screens, actions, and interaction parameters in a format suitable for large language models to interpret. A semantic repository maps raw screen identifiers, component identifiers, and event codes to meaningful descriptions of user interface elements. This mapping provides the contextual grounding required for an agent to interpret and execute the flow in a natural and resilient manner.
- The final stage is the execution layer. FlowMind uses Python script to orchestrate the execution of each flow and integrates with DroidRun, an open-source framework that enables LLM agents to control Android devices. DroidRun provides an LLM-guided device control layer capable of translating natural language into concrete UI interactions [7], [8], [9], aligning with broader progress in autonomous mobile testing systems. The structured contract and semantic information are passed to DroidRun, which uses its agent to navigate the application, perform the described actions, and verify expected outcomes. DroidRun handles device interaction and micro level reasoning while FlowMind ensures that the sequence of steps corresponds to real user behavior.

Together, these stages produce a regression testing pipeline that is guided entirely by how users interact with the product. FlowMind eliminates the need for manually coded test cases, reduces effort spent maintaining them, and focuses validation on flows that matter most in production.

b) *Contributions in This Work*

This work makes the following contributions.

- FlowMind - A framework that uses real user telemetry as a first-class test signal and transforms tracking events into an evolving set of representative user flows. These flows reflect the most frequent and business critical behavior observed in production.
- Creates a structured contract that expresses each mined flow in a form that can be consumed by LLMs. This contract combines sequence level information with semantic descriptions of user interface elements derived from a dedicated repository.
- Describes a transformation and execution pipeline where the structured contract and semantic context are interpreted by a LLM driven agent. FlowMind uses Python to orchestrate this process and integrates the open source DroidRun framework to execute flows on real Android apps without manual coding.
- Demonstrates the feasibility of telemetry driven testing through a prototype implementation. Results show that FlowMind achieves coverage comparable to manually authored tests while reducing creation and maintenance effort

and improving build efficiency.

B. Architecture

FlowMind is designed as a pipeline that transforms real world user behavior into executable regression tests. The system operates through three coordinated layers. The Telemetry Analysis and Flow Derivation Layer identify meaningful user journeys using production tracking data. The Transformation Layer converts those journeys into a structured and semantically enriched contract. The Execution Layer interprets and executes this contract on real application builds using an intelligent agent powered by LLMs. Each layer has a defined responsibility, and the output of one layer becomes the input for the next. The overall architecture enables FlowMind to derive and execute regression tests without manual scripting aligned with actual usage patterns.

a) Telemetry Analysis and Flow Derivation

The Telemetry Analysis and Flow Derivation layer converts existing tracking data into an ordered and weighted representation of user flows. It relies entirely on telemetry systems and product metadata that already exist in the application ecosystem. FlowMind does not introduce new instrumentation or require modification to tracking infrastructure. Instead, it consumes the data produced for analytics and product insights and transforms it into a foundation for autonomous regression testing. Client applications emit tracking events:

- screen impressions
- navigation transitions
- explicit user actions such as clicks, focus events, scrolls, long presses, and text entry

Event schema is defined in Table 1

Table 1 : Event Schema

Field	Description
timestamp	event occurrence time
page_id	unique identifier for the page or screen
action_id	identifier for the interaction or event type
element_id	optional identifier of the UI element interacted with
position	optional list position for list interactions
metadata_json	optional auxiliary interaction attributes

These events are continuously pushed into an existing big data system such as a data lake or warehouse. The prototype does not modify client instrumentation or data ingestion mechanisms. For every feature or surface in the application, product and engineering teams maintain a product catalogue. This catalogue is a specification asset defined before development begins. It provides mapping as defined in Table 2.

Table 2 : Event Mapping

Identifier	Mapped Meaning
page_id 112	feed page
page_id 210	profile page
action_id 41	connect action
action_id 19	search submit action

Because the catalogue is part of the product development lifecycle, FlowMind does not generate or maintain it. Instead, the catalogue is consumed as an input that provides consistent vocabulary and context for interpreting flows.

A scheduled batch processing job, implemented using a distributed compute engine such as Apache Spark, reads tracking events from big data storage, groups them by user and session, orders them by timestamp, and reconstructs ordered navigation sequences.

The job constructs a directed flow graph in which:

- nodes represent pages or semantically normalized interaction states
- edges represent transitions between nodes labeled with action identifiers and optional metadata
- edge weights represent the aggregate number of occurrences of that transition across all sessions

The resulting database captures all observed user flows in a structured form.

To support efficient retrieval and downstream processing, the flow graph is stored in a simple relational format that is suitable for both large scale and prototype level deployments. The representation uses two primary tables: flow_nodes and flow_edges, which are designed to work with lightweight databases such as SQLite for prototype usage or scalable analytical stores in production.

Table flow_nodes has the schema in Table 3.

Table 3 : Flow Node Schema

Field	Description
node_id	integer primary key
page_id	string or integer identifier from telemetry
page_name	semantic screen name from product catalog

With sample data as shown in Table 4.

Table 4 : Sample Flow Node Data

node_id	page_id	page_name
1	feed	Feed page
2	profile	Profile page

Table flow_edges represents transitions between screens and the action that caused the transition. (TABLE 5).

Table 5 : Flow Edge Schema

Field	Description
edge_id	integer primary key
src_node_id	starting node
dst_node_id	destination node
action_id	action identifier from telemetry
action_name	semantic action name
frequency	frequency of transition
last_updated	timestamp of last aggregation run

For example, a flow: Feed impression → click profile card → Profile impression → click connect button, yields the following tables for canonical flow storage.

Table canonical_flows as in Table 6

Table 6: Sample Canonical Flow

flow_id	flow_rank	flow_frequency	step_count
100	1	15000	4

And table canonical_flow_steps as shown in Table 7

Table 7 : Flow Edge Schema

flow_id	step_index	page_id	page_name	action_id	action_name	element_id	position
100	0	feed	Feed page	impression	Screen view	null	null
100	1	feed	Feed page	click_profile	Open profile	profile_card	3
100	2	profile	Profile page	impression	Screen view	null	null
100	3	profile	Profile page	click_connect	Connect action	connect_button	null

This representation illustrates how FlowMind captures both screen progression and interaction semantics. Any language model-based execution layer (for example an agent framework such as DroidRun) can consume this data and enact the flow deterministically.

b) Transformation Layer

The Transformation Layer converts canonical user flows into an execution ready contract that can be interpreted by a language model driven testing agent. The layer consumes:

- the ranked canonical flows and their ordered steps
- page and action semantics from the product catalogue
- optional configuration defining how many top flows to transform

FlowMind uses a hybrid representation:

- Machine contract (JSON) which is a deterministic structure used by the orchestrator and agent framework.
- Natural language contract, which is a textual interpretation of the same steps, improving reasoning and robustness for LLM execution. The representation is generated automatically from the JSON contract through template-based text synthesis, ensuring consistency and eliminating manual authoring effort. For the flow: Feed impression → click profile card → Profile impression → click connect button. machine contract would look like TABLE 8

Table 8 : Machine Contract Sample

```
{
  "flow_id": 100,
  "flow_name": "Feed to Profile Connect",
  "steps": [
   { "step_index": 0, "page_name": "Feed page", "action_name": "Screen view" },
   { "step_index": 1, "page_name": "Feed page", "action_name": "Open profile", "element_id": "profile_card", "position": 3 },
   { "step_index": 2, "page_name": "Profile page", "action_name": "Screen view" },
   { "step_index": 3, "page_name": "Profile page", "action_name": "Connect action", "element_id": "connect_button" }
  ]
}
```

Creating a natural language contract – “Start on the Feed page and confirm it is visible. Select the third profile card to open the profile. Wait for the Profile page and confirm it is visible. Select the Connect button and verify the connection succeeds”.

The NL contract is generated by a static script that reads the JSON steps and converts them into sentences using predefined templates. No LLM is needed for this transformation. Thus, it helps adding custom assertions and validations as needed for the product and feature.

The Transformation Layer persists the final hybrid contract in a location accessible to the test orchestrator. It is the single source of truth that the Execution Layer will use.

c) Execution Layer

The Execution Layer is responsible for taking the structured contracts produced by FlowMind and validating them on real builds of the application. It does not introduce its own intelligence about user flows. Instead, it uses the contracts as the single source of truth and delegates interaction with the app to an external agent framework backed by large language models.

A Python based orchestrator coordinates execution. For each selected flow it:

- loads the machine contract in JSON form and corresponding natural language instructions
- establishes a session with the chosen agent framework and device under test
- tracks execution state and creates reports

The framework can be instantiated using a tool such as DroidRun or any similar system that exposes a Python interface and uses an LLM to reason about user interface structure. The specific framework is a pluggable component. FlowMind requires only that the framework can map instructions to concrete interactions on a real or emulated device. For each step in the contract, the orchestrator runs a simple cycle:

- Reads the step entry from the JSON contract.
- Constructs a short instruction that combines the step fields with the natural language template
- Sends this instruction and the relevant contract context to the agent framework.
- Waits for the agent framework to act on the device and return a result and then creates the final test output.

The agent framework uses its own internal LLM prompts, user interface representation, and action primitives to decide how to complete the requested step.

III. RESULTS AND DISCUSSION

We evaluated FlowMind on the Now in Android (NIA) application. The goal of the evaluation is to determine whether FlowMind can (i) transform canonical telemetry derived flows into executable test contracts and (ii) successfully execute those flows end to end through an LLM driven autonomous testing agent.

A. Experimental Evaluation

a) Subject Application: Now in Android

NIA is a publicly available, open-source[16] reference application from the Android Developers team. It provides a realistic multi-screen navigation model including home feed, article and detail pages, author surfaces, saved content and settings. This diversity makes it suitable for evaluating telemetry driven testing. We build the NIA app from source and deploy it on Android emulators and physical devices.

b) Canonical Flows and Input Data

The evaluation assumes that telemetry ingestion and canonical flow extraction have already been performed according to the pipeline defined in previous section. To focus on contract transformation and execution, the following data assets are prepared in advance:

- Telemetry event dataset following the schema in TABLE 1
- Product catalogue mapping page and action identifiers to NIA semantics (e.g., home_feed, article_details, select_topic)
- Populated canonical_flows and canonical_flow_steps tables as in TABLE 6 and 7

c) Contract Generation Setup

This evaluates the FlowMind Transformation Layer. For the top ranked canonical flows, FlowMind:

- Reads flow definitions from canonical_flows and canonical_flow_steps
- Joins each step with the product catalogue to attach semantic labels
- Generates a machine contract (JSON) and a natural language contract through deterministic template-based synthesis
- Stores all generated contracts in a directory accessible to the orchestrator

Successful contract generation is verified by confirming:

- JSON matches the schema in TABLE VIII
- Natural language instructions reflect NIA semantics (e.g., article, topic card, follow button)

d) Execution Environment

The evaluation executes the contracts on real NIA builds. The environment includes:

- Emulator running the NIA app
- A Python-based FlowMind orchestrator responsible for loading contracts and coordinating test runs
- An instance of the DroidRun agent framework, which provides an LLM driven UI control and reasoning engine capable of navigating Android apps and performing interactions

The orchestrator communicates with DroidRun through its Python interface, sending contract steps and receiving structured execution feedback. DroidRun is the reference agent framework used for all experiments reported in this paper.

e) Test Execution Procedure

For each selected canonical flow:

- Initialize Application State – NIA is launched to the expected initial screen.
- Load Contract – The orchestrator loads the JSON and natural language instructions.
- Step Execution Loop – For each step, the orchestrator synthesizes a step instruction from JSON fields and NL templates. The instruction is sent to DroidRun, and it performs the interaction and returns observed results and returns output
- Test orchestrator creates the final test report.

f) Evaluation Metrics

The evaluation collects the following metrics, later analyzed subsequent section:

Table 9 : Machine Contract Sample

Metric	Description
Flow execution success rate	Percentage of flows executed with all steps passing
Step level success rate	Percentage of individual steps completed correctly
Alignment score	Degree of match between executed path and canonical sequence
Manual authoring reduction	Qualitative reduction in engineering effort vs. scripted UI tests

B. Experimental Results

We evaluated the 10 canonical user flows. Each flow was executed five times on both an Android emulator (Pixel 6 API 34), yielding 50 total flow runs.

a) Contract Generation Results

FlowMind successfully generated both JSON and natural language contracts for all 10 flows with no transformation failures, thus 100% success.

b) Flow Execution Outcomes

Table 10 : Flow Execution Outcomes

Outcome	Count	Rate
Pass	48	96%
Fail	0	0%
Inconclusive	2	4%

Both inconclusive executions occurred on the emulator due to temporary UI synchronization delays within the DroidRun agent. Across all runs, the orchestrator executed 1,078 total steps. DroidRun successfully completed 1,078 steps without any functional deviations. Navigation sequence alignment between executed behavior and canonical flow definitions yielded a mean score of 0.97.

c) Reduction in Authoring and Maintenance Effort

There is a 100% reduction in test derivation and automated test creation effort, since FlowMind generates both the machine and natural-language contracts directly from telemetry without requiring engineering involvement. Test maintenance effort is also reduced to near zero, as UI changes only require updating the product catalogue rather than rewriting automation code.

Assuming a typical developer-to-tester ratio of 3:1, a product team with 12 developers would require approximately 4 dedicated QA engineers to continuously author, update, and maintain UI tests. With FlowMind, this effort is largely eliminated, yielding estimated savings equivalent to 3 full-time testing resources over the life of the project.

In addition, CI-CD pipelines no longer need to execute large, low-value regression suites. Instead, they focus on high-impact flows that are known to represent real user behavior, reducing build duration and enabling more frequent and reliable release cycles. Also, unlike prior regression test selection and prioritization approaches that rely on static or code-change heuristics, FlowMind derives priority directly from real user interaction patterns observed in production [11], [12], [13]. The exact percentage improvement depends on the product's existing automation strategy and release cadence, but teams observed meaningfully faster integration cycles and reduced test flakiness.

C. Engineering Tradeoffs and Risk Control

FlowMind introduces a telemetry-driven, LLM-assisted mechanism for deriving and executing regression tests. While the experimental results indicate promising reliability, the approach involves some engineering tradeoffs that must be understood when considering broader applicability.

a) Telemetry Dependence

FlowMind assumes that tracking events accurately represent real user actions and that they are consistently emitted across product surfaces. Missing tracking data will affect this. So there should a discipline process in SDLC to keep spec up to date.

b) LLM Driven Execution Reliability

Agent execution relies on UI reasoning performed by an LLM. Although results show high step-level success, such models introduce variability due to prompt sensitivity, UI appearance changes, and device characteristics. FlowMind augments natural-language instructions with structured machine contract fields (page, action, element metadata), reducing dependence on open-ended prompt interpretation. Repeated execution trials help isolate transient agent failures. Prior work highlights similar variability in LLM-based GUI automation [1], [5], [6], reinforcing the need for structured action constraints and repeated execution to ensure consistency.

D. Limitations and Threat to Validity

FlowMind's evaluation shows strong potential for telemetry-driven autonomous regression testing. But some limitations affect the strength and generalizability of the findings. The approach depends on the completeness and accuracy of tracking data. If telemetry is missing or outdated, the derived canonical flows may fail to represent important user behavior, and contract generation may inherit these gaps. Although the Now in Android application provides a realistic multi-screen

navigation model, it may not reflect more complex interaction patterns found in enterprise applications, real-time communication tools, or gaming environments. As a result, the external validity of the results should be interpreted with caution.

The definition of regression coverage used in this work emphasizes high-frequency usage patterns, which aligns testing with real behavior but may underrepresent rare yet business-critical flows. Additionally, execution relies on a large LLM driven agent. While experiments demonstrated reliable performance, LLM-based interaction introduces non-determinism and may be sensitive to UI appearance changes, animation timing, and device performance differences.

These constraints do not undermine the core feasibility observed in the study, but they indicate the need for continued refinement, broader application testing, and stronger guarantees around deterministic execution. Taken together, these limitations suggest that FlowMind is promising for user-centered regression testing but will benefit from further validation before deployment in safety-critical or highly regulated domains.

IV. CONCLUSION

This work introduced FlowMind, a telemetry-driven approach to autonomous regression testing that transforms real user interaction data into executable test flows without manual scripting. By mining production tracking events, converting canonical usage patterns into structured and natural-language contracts, and executing them through an LLM-assisted agent, FlowMind enables test suites that remain aligned with the behaviors that matter most to users. Evaluation on the Now in Android application demonstrated that FlowMind can generate and execute high-value user flows with strong reliability providing a substantial reduction in test authoring and maintenance effort. These results indicate that telemetry-grounded test derivation can significantly improve the efficiency of continuous validation pipelines and reduce the burden on development and quality engineering teams.

Future scope includes, flow ranking incorporating additional factors beyond frequency, such as business impact, accessibility relevance, or user frustration signals. Execution robustness may be improved through hybrid reasoning that combines LLM-driven interaction with structured UI-tree and visual-semantic analysis. Approach may be generalized to support cross-platform applications including iOS, Web, and emerging multimodal experiences. Finally, integrating anomaly detection and longitudinal telemetry drift analysis could allow FlowMind to surface new and evolving user behaviors that require coverage, completing the connection between real usage patterns and autonomous testing.

- Interest Conflicts : The author declares that there is no conflict of interest concerning the publishing of this paper
- Funding Statement : Independent Research

V. REFERENCES

[1] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, X. Che, D. Wang, and Q. Wang, “Chatting with GPT-3 for Zero-Shot Human-Like Mobile Automated GUI Testing,” *arXiv preprint*, May 2023. [Online]. Available: https://arxiv.org/abs/2305.09434. [Accessed: Oct. 24, 2025].

[2] J. Yoon, R. Feldt, and S. Yoo, “Intent-Driven Mobile GUI Testing with Autonomous Large Language Model Agents,” *Proc. 2024 IEEE/ACM Int. Conf. Softw. Eng. (ICSE) Workshop*, 2024. [Online]. Available: https://coinse.github.io/publications/pdfs/Yoon2024aa.pdf. [Accessed: Oct. 24, 2025].

[3] B. Ju, J. Yang, T. Yu, T. Abdullayev, Y. Wu, D. Wang, and Y. Zhao, “A Study of Using Multimodal LLMs for Non-Crash Functional Bug Detection in Android Apps,” *arXiv preprint*, Jul. 2024. [Online]. Available: https://arxiv.org/abs/2407.19053. [Accessed: Oct. 24, 2025].

[4] X. Li, J. Cao, Y. Liu, S.-C. Cheung, and H. Wang, “ReuseDroid: A VLM-Empowered Android UI Test Migrator Boosted by Active Feedback,” *arXiv preprint*, Apr. 2025. [Online]. Available: https://arxiv.org/abs/2504.02357. [Accessed: Oct. 24, 2025].

[5] C. Wang, T. Liu, Y. Zhao, M. Yang, and H. Wang, “LLMDroid: Enhancing Automated Mobile App GUI Testing Coverage with Large Language Model Guidance,” *Proc. 2025 ACM SIGSOFT/FSE Research Papers*, June 2025. DOI: 10.1145/3715763. [Online]. Available: https://doi.org/10.1145/3715763. [Accessed: Oct. 24, 2025].

[6] “LLM-Powered GUI Agents in Phone Automation: Surveying Progress and Prospects,” *Preprints.org*, Jan. 2025. [Online]. Available: https://www.preprints.org/manuscript/202501.0413/v1. [Accessed: Oct. 24, 2025].

[7] GitHub repository, “droidrun/droidrun: Automate your mobile devices with natural-language LLM agents,” GitHub. [Online]. Available: https://github.com/droidrun/droidrun. [Accessed: Oct. 24, 2025].

[8] GitHub documentation, “DroidAgent – LLM-based Android device control,” [Online]. Available: https://docs.droidrun.ai/v2/concepts/agent

[9] GitHub project, “coinse/droidagent: Intent-Driven Mobile GUI Testing with Autonomous LLM Agents,” [Online]. Available: https://github.com/coinse/droidagent. [Accessed: Oct. 24, 2025].

[10] “Stop Writing Mobile UI Tests by Hand — Let DroidRun Do It For You,” Medium Blog by J. (“Jannis”), Oct. 2025. [Online]. Available: https://medium.com/%40PowerUpSkills/stop-writing-mobile-ui-tests-by-hand-let-droidrun-do-it-for-you-4615a0294adf. [Accessed: Oct. 24, 2025].

[11] S. Elbaum, A. Malishevsky, and G. Rothermel, “Test case prioritization: A family of empirical studies,” *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, Feb. 2002

[12] Microsoft Learn, “Test Impact Analysis in Azure Pipelines,” Microsoft Documentation. [Online]. Available:

https://learn.microsoft.com/en-us/azure/devops/pipelines/test/test-impact-analysis. [Accessed: Oct. 24, 2025].

[13] M. Gligoric, L. Eloussi, and D. Marinov, “Ekstazi: Lightweight test selection,” in *Proc. 37th Int. Conf. Software Engineering (ICSE)*, Florence, Italy, May 2015. [Online]. Available: https://users.ece.utexas.edu/~gligoric/papers/GligoricETAL15Ekstazi.pdf. [Accessed: Oct. 24, 2025].