
Assessing and Understanding AI Coding Agents

Kevin Moran, Ph.D.

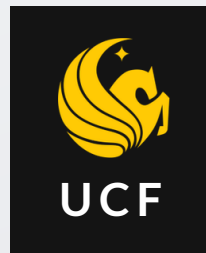
Assistant Professor, CS

Director of the SAGE Research Lab

University of Central Florida

Senior Design Kickoff

Fall 2026 – Spring 2027



About Myself



Kevin Moran, Ph.D.

Assistant Professor

Department of Computer
Science

kpmoran@ucf.edu · kpmoran.com



RESEARCH INTERESTS

- Software Engineering, Maintenance & Evolution
- Artificial Intelligence for Software Engineering
- Software Security

TEACHING

- Software Engineering (CEN 5016)
- User Interface Design & Development (SWE 632)
- Web Development (SWE 432)

SELECTED AWARDS

- **NSF CAREER Award** (2025)
- **ACM SIGSOFT Early Career Researcher Award** (2024)
- **UCF “Reach for the Stars” Award** (2026) — the university’s highest pre-tenure faculty honor
- **Rising Star Award**, MobileSoft (2023)
- Multiple **ACM SIGSOFT Distinguished Paper Awards**

Member of the UCF Cyber Security and Privacy Cluster (CyberSP).

The SAGE Lab & Our Industry Ties



The Software Automation, Generation & Engineering Research Lab

sagelab.io · 6 Ph.D. students across UCF and GMU



Safwat



Atish



David



Sparsh

AI for Software Engineering

Code and multimodal models turned into developer tools that hold up under real use.

Agentic Software Engineering

What coding agents can do, where they break, and how we would know either way.

Automated UI Analysis

Teaching machines to read a screen the way a user does — pixels, hierarchy, intent.

\$6.7 million

in research funding to date, from government and industry sources.

SPONSORS & INDUSTRY PARTNERS



WHERE OUR PEOPLE GO



Arun Krishna Vajjala

ML Research Engineer
Apple



SM Hasan Mansur

Tenure-Track Professor
Stephen F. Austin State



Junayed Mahmud

Research Engineer
Deloitte



Samar Karanch

M.S. student
UC San Diego



Kristen Goebel

Ph.D. student
Oregon State



Jasmine Obas

Software Engineer
Bonne Terre Labs



Yule Zhang

Software Engineer
UPS

Four of these started as undergraduates.

Past Senior Design Success

LadyBug github.com/LadyBugML/ladybug

A GitHub bot that localizes bugs in Android apps by combining **UI interaction traces** with text retrieval. A developer files an issue with a reproduction trace; LadyBug replies with a ranked list of the files most likely to contain the defect.

LadyBugML / ladybug Public	
★ 7 stars : 2 forks 15 branches 387 commits MIT licence	
Ladybug is an automated bug localization tool for Android applications that leverages advanced GUI techniques to significantly improve accuracy over traditional text-retrieval methods.	
SamarKaranch Merge pull request #103 from LadyBugML/print-statement	
■ .github/workflows	Update Requirements and CI/CD pipeline
■ backend	Add embeddings print
■ probot	Fix duplicate comments on initialization
▢ Dockerfile.backend	Added support to cuda
▢ docker-compose.yml	update docker & requirements to fix deployment
▢ README.md	Test change for CI/CD

Seven contributors, 387 commits — and the merge at the top of the log is one of the seniors.

WHAT IT BECAME

LadyBug: A GitHub Bot for UI-Enhanced Bug Localization in Mobile Apps — a Tool Demonstration paper at ICSME 2025, the IEEE International Conference on Software Maintenance and Evolution.

Mahmud, Chen, Achille, Alvarez-Velez, Bansil, Ijeh, Karanch, De Silva, Chaparro, Marcus, Moran.
arxiv.org/abs/2508.05085

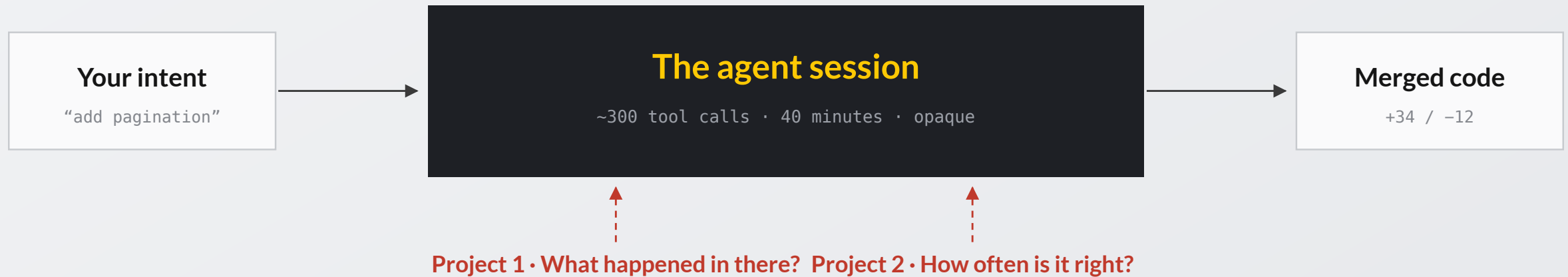
Six of the eleven authors were the Senior Design team.



Two of them flew to Auckland, New Zealand to present it.

Two Projects for Senior Design

AI coding agents now write a real share of production software. Two basic instruments are still missing: a way to *see* what a single agent run actually did, and a way to *measure* how often agents get it right on the software people touch.



PROJECT 1

Visualizing AI Coding Agent Traces

A tool that makes a single agent session legible — so a developer can see what the agent did, and why.

The Problem

FRIDAY, 4:47 PM

You ask the agent to add pagination to the orders endpoint. It runs for 40 minutes. The test suite is green. You skim the diff and merge.

MONDAY, 9:10 AM

Support tickets. Three customers report orders in their history that were never theirs. The tenant filter is gone.

MONDAY, 9:11 AM

You open the diff, then the logs. Neither one tells you what the agent was doing when it removed that filter.

The agent's reasoning is gone. Only its output survives.

What the Developer Actually Sees

```
git diff - 6 files changed, +34 -12
```

```
--- a/api/orders/query.py
+++ b/api/orders/query.py
@@ -41,9 +41,11 @@
 def list_orders(user, page, size):
-    q = Order.where(tenant=user.tenant)
+    q = build_query(user)
+    q = q.offset(page * size)
+    q = q.limit(size)
     return q.all()

--- a/api/orders/routes.py
@@ -18,3 +18,5 @@
+    page = int(request.args.get("page", 0))
+    size = min(int(request.args.get("size", 25)), 100)

... 4 more files: 3 test files, 1 helper.
```

```
session.jsonl - 918 events
```

```
[0088] tool_use  bash      rg "def list_orders"
[0089] tool_use  read_file  api/orders/routes.py
[0090] tool_use  read_file  tests/test_orders.py
[0091] tool_use  read_file  api/orders/query.py
[0092] tool_use  grep      "tenant="
[0093] tool_use  read_file  api/common/qb.py
[0112] tool_use  edit_file  api/common/qb.py

[0113] tool_use  bash      pytest -q
[0114] result    28 passed, 0 failed
[0115] tool_use  read_file  tests/test_orders.py
[0116] tool_use  edit_file  api/orders/query.py
[0117] tool_use  bash      pytest -q
[0118] result    28 passed, 0 failed

... 801 more lines ...
```

The diff is small and looks reasonable. **build_query()** is where the tenant filter went — edited at event 112, in a file the diff barely mentions, for a reason that appears in neither artifact.

Why Neither Artifact Helps

THE SESSION — 918 EVENTS, 302 TOOL CALLS



The log has **every step** and no structure.

The diff has **structure** and no steps.

Nothing connects a line of code to the decision that produced it.

What You Would Build



When was this line written, and what was the agent trying to do?

Where did the agent go in circles or undo its own work?

What did it read before it made the decision?

Which changes were never covered by a test?

Project 1 – Scope

THE CORE DELIVERABLE

- **A trace viewer** that loads a real agent session and renders a navigable timeline.
- **Line-level provenance** — click a line of code, get the step and the stated reason.
- **Loop and dead-end detection**, surfaced automatically.

We supply real JSONL session traces from Claude Code, Codex and similar agents.

STRETCH GOALS

- **Attach to a live session** and watch the agent work in real time.
- **Shareable session reports** a reviewer reads instead of the diff.
- **An editor extension**, so provenance appears where you already work.
- **A user study with real developers** — a publishable paper with your name on it.

PROJECT 2

Building a Leaderboard for AI Agents Applied to Fixing UI-Centric Bugs

A benchmark anyone can submit a coding agent to, with a public leaderboard and the statistics behind every score.

Automated Program Repair

Techniques that generate a patch for a known bug, and check it against the project's tests.

What nobody has measured: how well these techniques — and the coding agents that succeeded them — work on ***user-facing software***, where the bug is something a person sees on a screen.

The Work Is Published and Public

You would not be starting from a blank page. The benchmark, the harness, the generated patches and the codebooks are all released – and the paper that motivates the project is peer reviewed.

SageSELab / **UI-Program-Repair**

Public

DOI 10.5281/zenodo.21521940

Available

Functional

Reusable

DroidFixBench — Artifact

The research artifact for “Automated Program Repair of UI-centric Android Bugs: How Far are We?” — the DroidFixBench benchmark, the source for all five APR tools, the complete set of generated patches, the manual-analysis codebooks and taxonomy, and a Docker harness that runs each tool over the benchmark.

● Java 77.4%

● Python 21.4%

29 commits

MIT licence

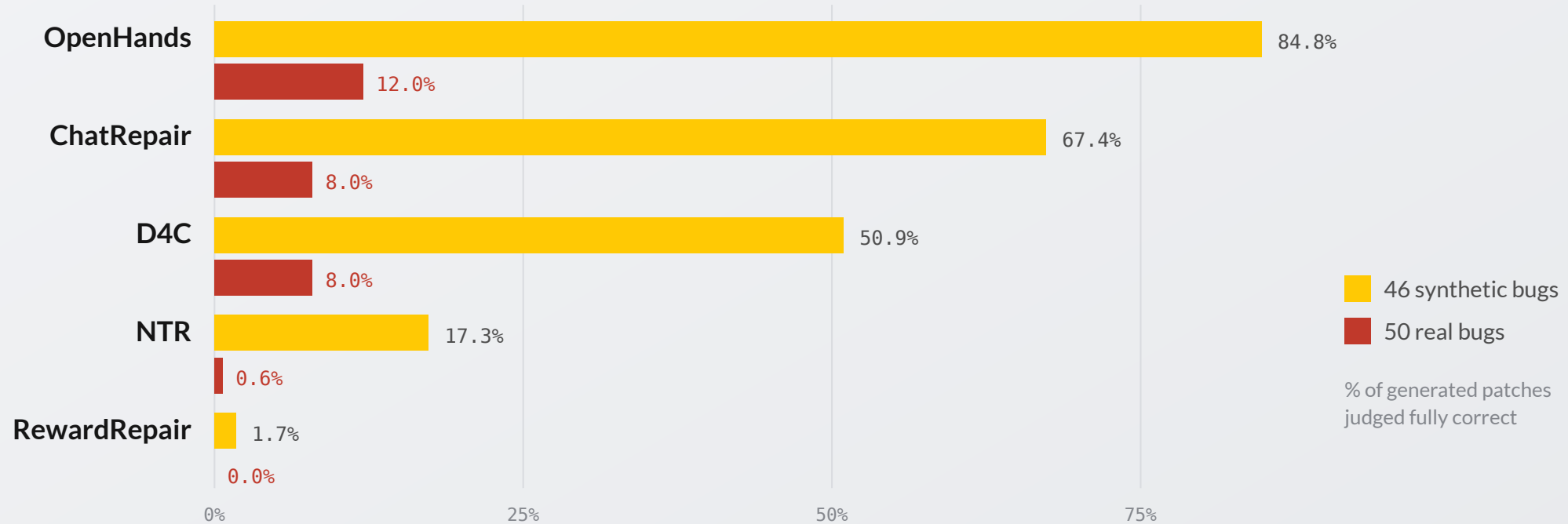
github.com/SageSELab/UI-Program-Repair — 96 bugs, five tools, every patch we generated, and the nine-category codebook your platform will score against.

Proc. ACM Softw. Eng. 3, ISSTA, Article 181 (October 2026).

doi.org/10.1145/3832272

What We Found — ISSTA 2026

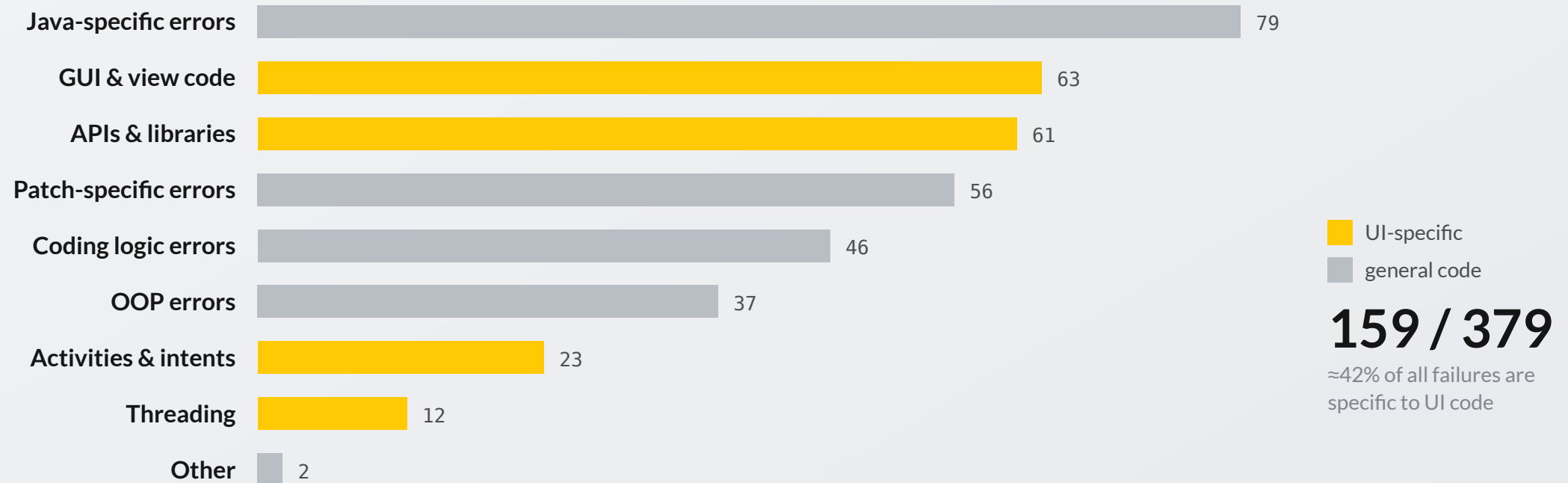
We built **DroidFixBench** — 50 real bugs mined from open-source Android apps and 46 synthetic ones — and ran five repair techniques across all 96.



Across all five tools on the 50 real bugs, **3.3% of patches were correct** and **67% did not compile**. On synthetic bugs the same tools look excellent. The gap is the finding.

Why UI Bugs Are Hard for Repair Tools

We hand-coded a statistically significant sample of **379 failed patches** into nine categories.



35 patches referenced GUI component IDs that do not exist • **37** called Android methods that do not exist • **18** invented Activities. The models hallucinate the interface they cannot see.

A Study Is a Snapshot

Frozen in time

Five tools, fixed models, one moment in 2025. A new coding agent ships roughly every month.

Graded by hand

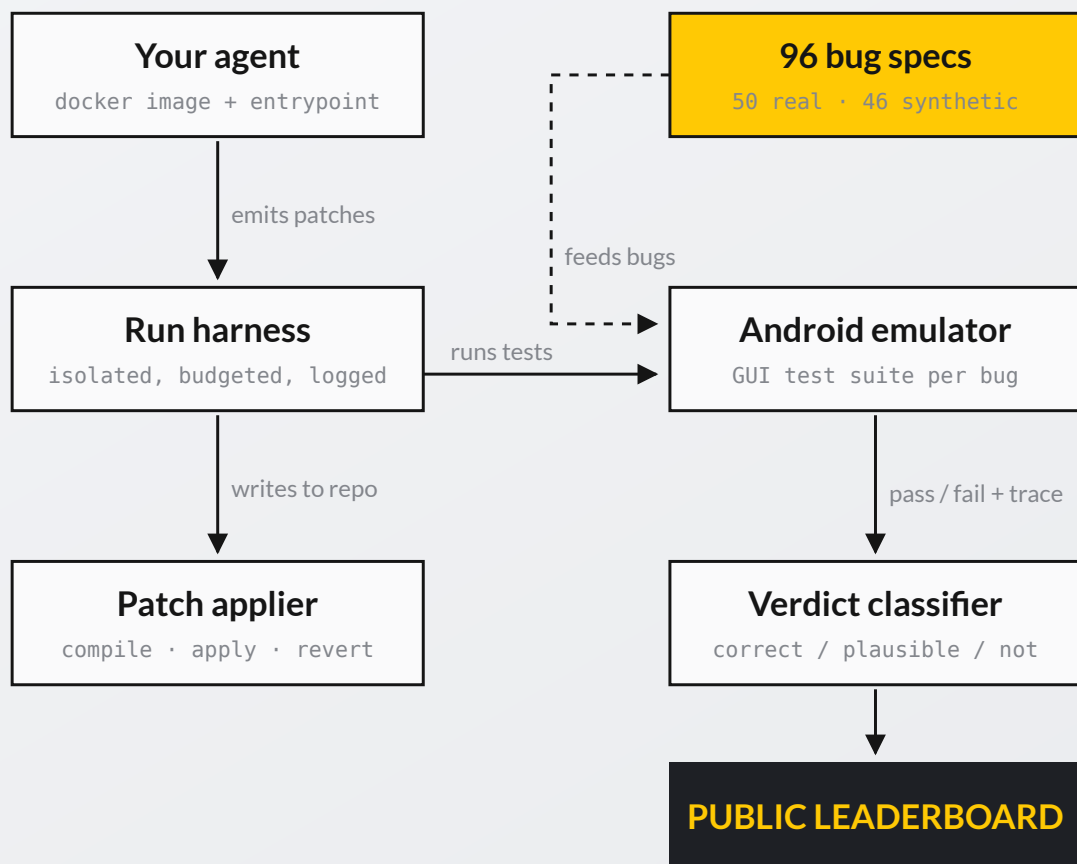
Two years of work, and 379 patches read one at a time by human annotators.

Not the agents you use

By design we measured classical repair first. Claude Code, Codex and Gemini CLI on these 96 bugs remain unmeasured.

So the project is not another study. It is the platform that keeps the question answered.

What You Would Build



Everything in the left column already exists in our repo. You build the loop, the scoring, and the front end.

THE LEADERBOARD

MOCKUP · EXAMPLE DATA

#	AGENT	REAL	SYNTH	\$ / FIX
1	agent-alpha	21.0%	89.1%	\$1.84
2	agent-beta	18.0%	91.3%	\$4.10
3	team-gamma	14.0%	84.8%	\$0.62
4	OpenHands	12.0%	84.8%	—
5	ChatRepair	8.0%	67.4%	—

Rows 4 and 5 are our published results — the seed of the board. The rest shows what submissions would look like.

The rank is the hook. The **drill-down is the contribution**: per-bug outcomes, real versus synthetic, failures classified against our nine-category codebook, and the cost, tokens and wall-clock of every run.

Project 2 – Scope

THE CORE DELIVERABLE

- **A submission pipeline** that takes a containerized agent and runs all 96 bugs.
- **Automated verdicts** – compile, run the GUI tests, classify correct / plausible / not.
- **A leaderboard with real depth** behind every number.

github.com/SageSELab/UI-Program-Repair

STRETCH GOALS

- **Open the board publicly** and run the first call for submissions.
- **Auto-classify failures** with a model judge validated against our human codebook.
- **Extend beyond Android** – the same harness shape fits web UI bugs.
- **Co-author the tool paper.** Benchmarks with real infrastructure get cited for years.

Choosing a Project

PROJECT 1

Visualizing Agent Traces

Front-end heavy. Data visualization, developer tooling, and real user research if you want it. You build the instrument every engineer using an agent is currently missing.

PROJECT 2

The DroidFixBench Platform

Infrastructure heavy. Docker, Android emulators, CI at scale, and a public web front end. You build the scoreboard this field gets judged on.

Both projects end with an artifact that does not currently exist, released publicly under the lab's name — and yours.

kpmoran@ucf.edu · sagelab.io · github.com/SageSELab

[Download these slides as a PDF](#)



THESE SLIDES
+ the PDF