Profile Coverage: Using Android Compilation Profiles to Evaluate Dynamic Testing

Jakob Bleier

Felix Kehrer

Jürgen Cito

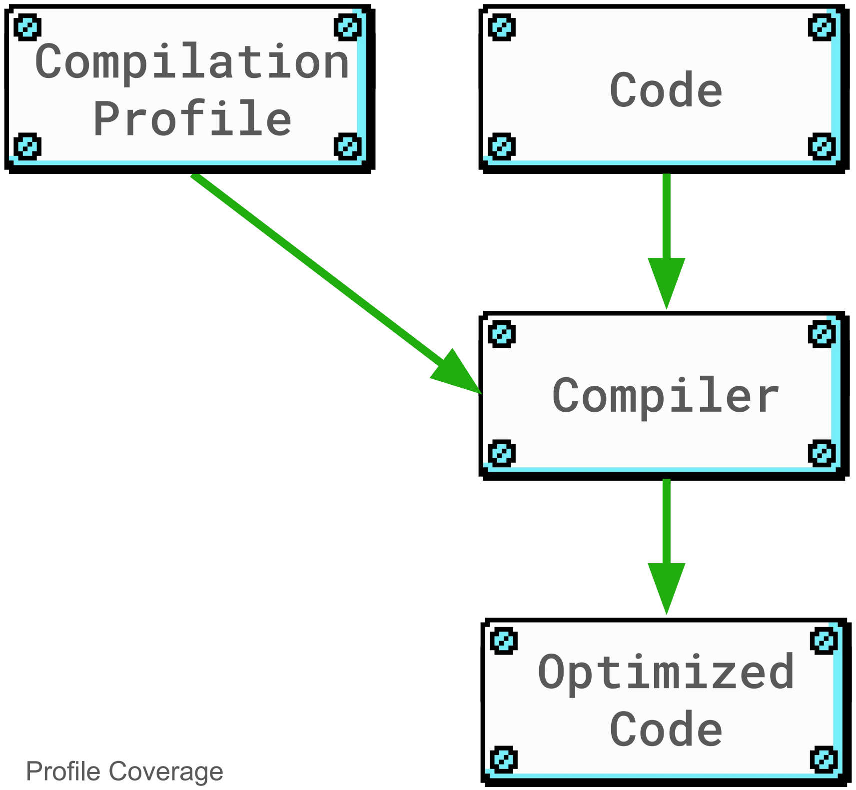
Martina Lindorfer

TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

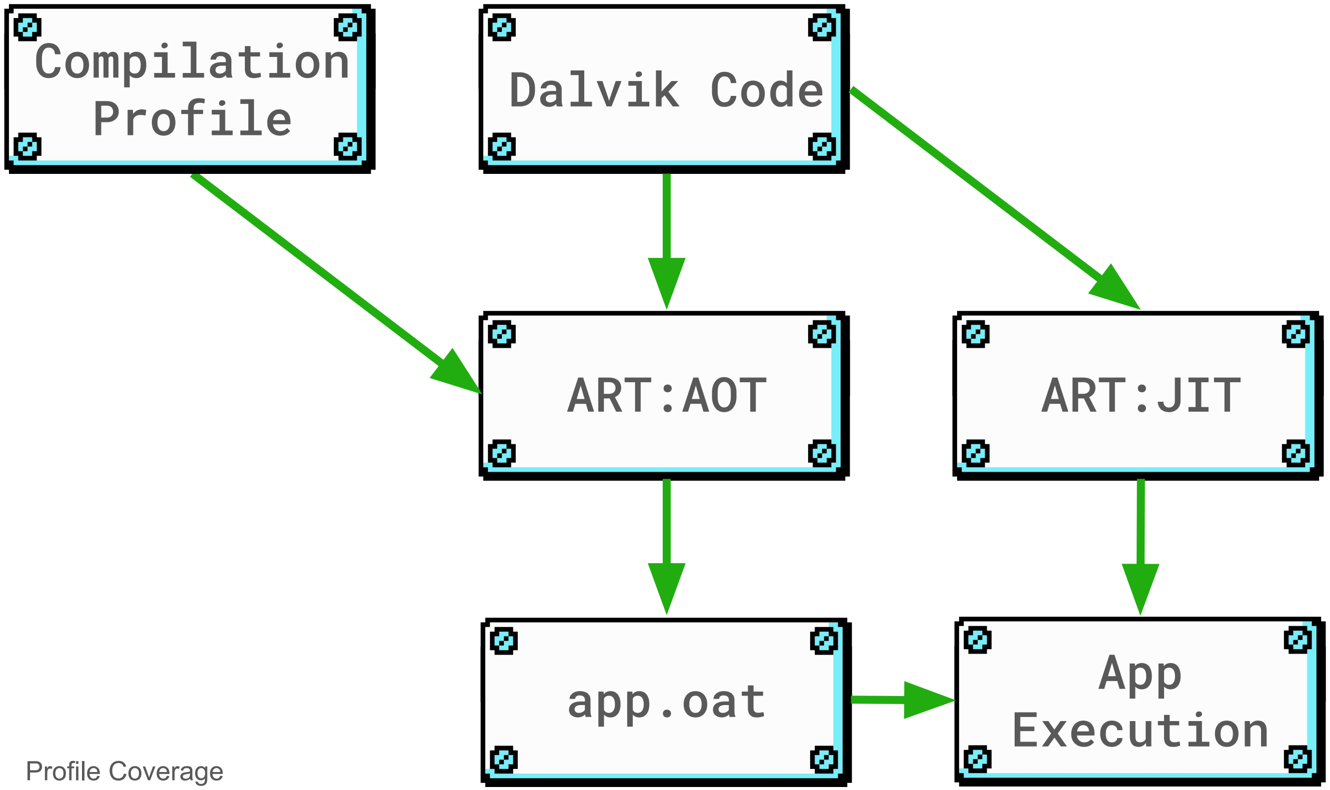
ASE 2025

Motivation

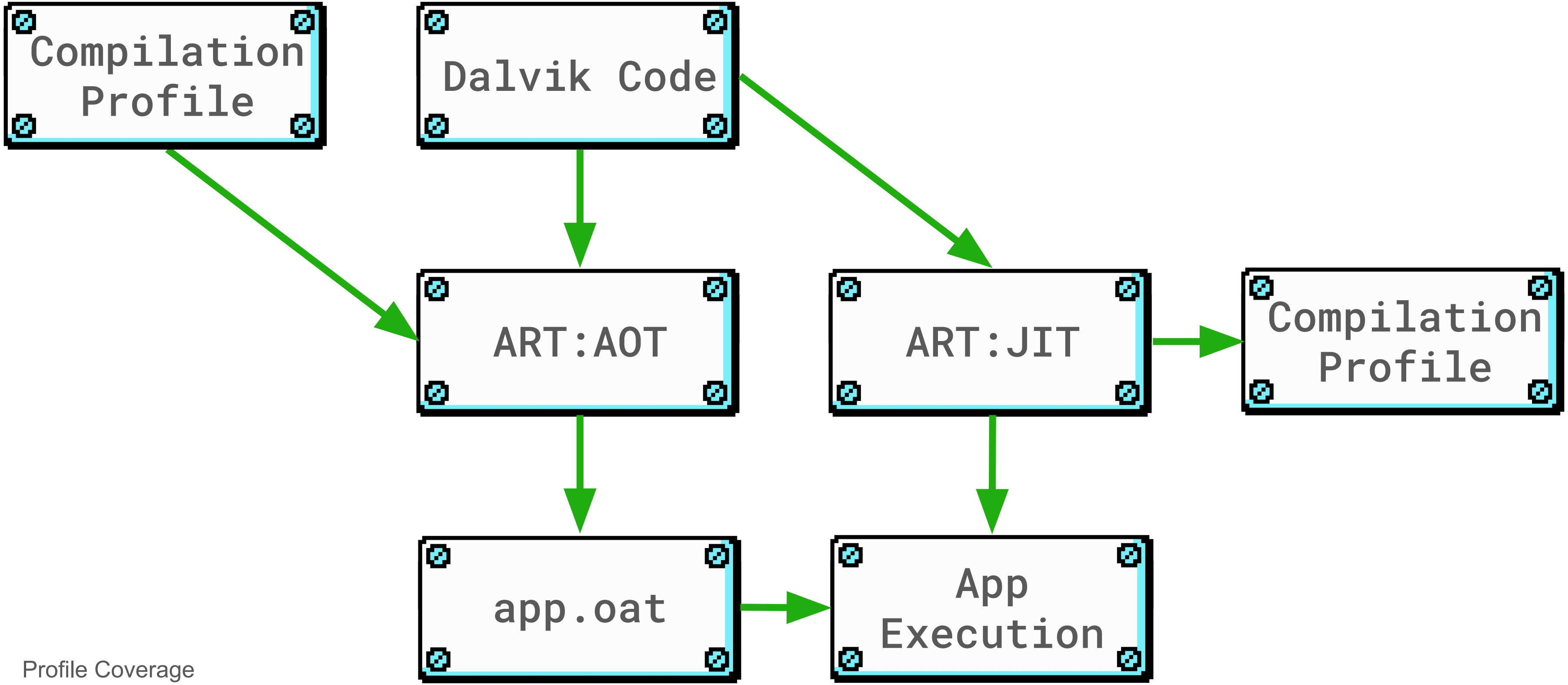
Compilation Profiles



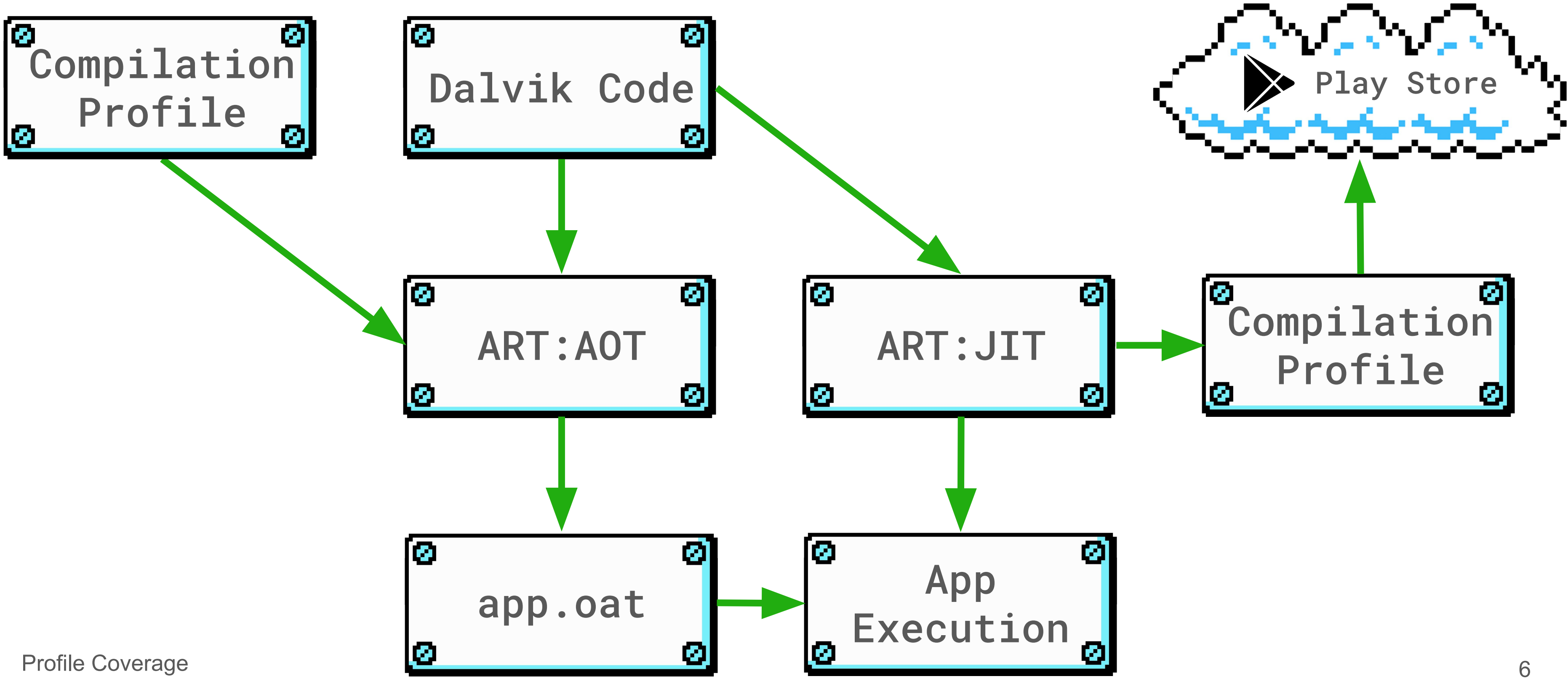
Compilation Profiles on Android



Compilation Profiles on Android



Cloud Profiles on Android



Cloud Profiles: Readily Available

Collected apps and their Cloud Profiles:

- Daily for over 7 weeks
- For top 1.000 apps

And found:

- Available for virtually all apps
- Overlap with Baseline Profiles <30%
- Represent aggregated usage information

Aggregated Usage Data to Evaluate Dynamic Testing

$$\text{Profile Coverage} = \frac{|\text{Methods}_{\text{Executed}} \cap \text{Methods}_{\text{Profile}}|}{|\text{Methods}_{\text{Profile}}|}$$

Tracing Profile Coverage

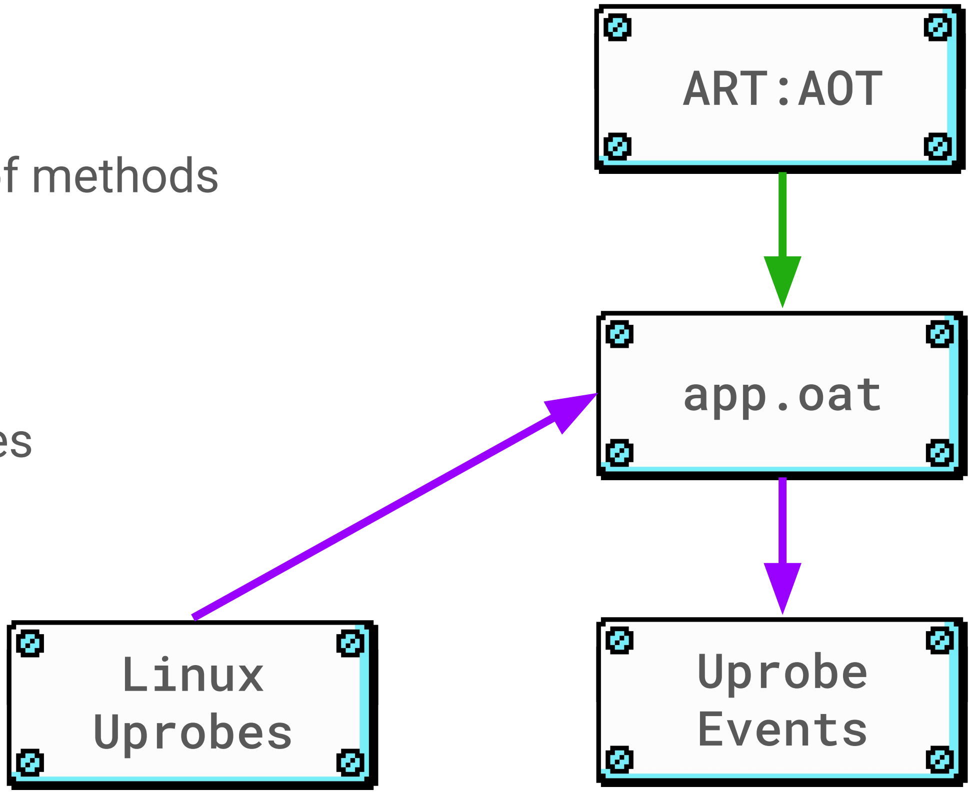
Needed a tool that:

- Is able to trace thousands of methods
- Does not repackage app
- Does not recompile system
- Works with emulator
- Works with hardware devices

Tracing Profile Coverage

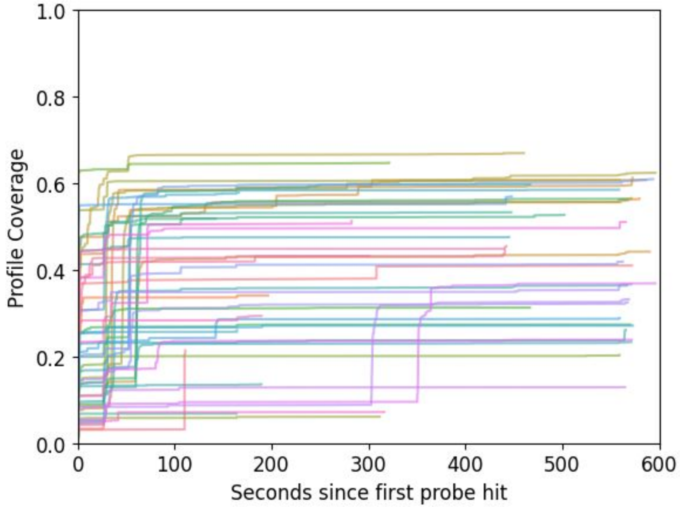
Needed a tool that:

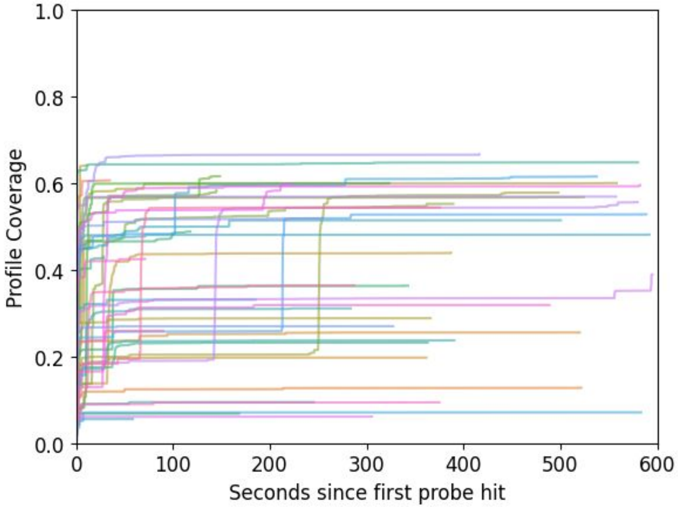
- Is able to trace thousands of methods
- Does not repackage app
- Does not recompile system
- Works with emulator
- Works with hardware devices



Evaluation

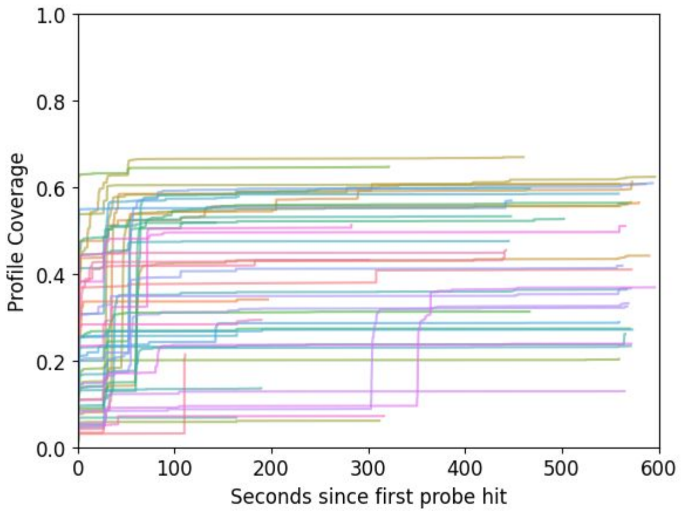
We exercised 50 apps with **Monkey**, **DroidBot**, **Fastbot2**, and a “No Interaction” **baseline**.



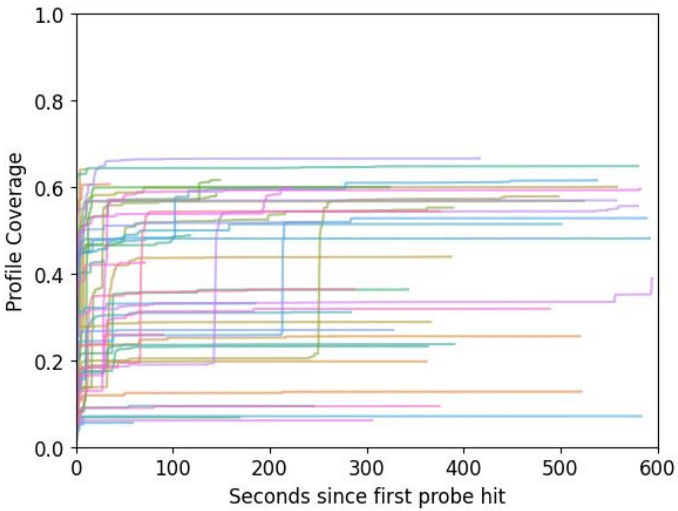


Evaluation

We exercised 50 apps with **Monkey**, **DroidBot**, **Fastbot2**, and a “No Interaction” **baseline**.



Monkey

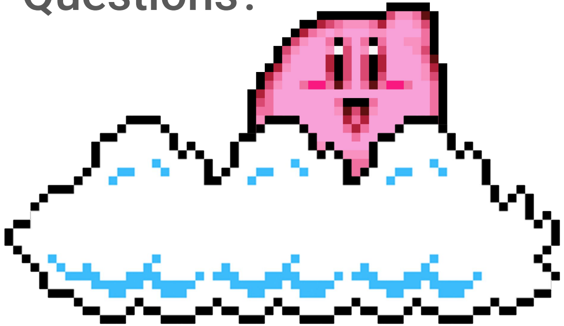


Fastbot2

Profile Coverage

- Empirical exploration of Cloud Profiles, including tooling to easily download them.
- ProfTrace, an uprobe-based tracing approach that works without changes to apps or the system.
- Evaluation of Profile Coverage on 3 test input generation tools and comparison to code coverage.

Questions?



TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Code available: https://github.com/SecPriv/android-profile-tracing