

# This Year in Uber's AI-Driven Developer Productivity Revolution

Uber



Adam  
Huda



Ty  
Smith

# Uber's Scale

**\$160** billion

Annualized run-rate gross bookings

**156** million

Monthly active platform consumers

**10,000**

+ Cities

**30** million

Trips per day

**7.4** million

Monthly active drivers and couriers globally

**70**

Countries

# Platform Leverage

## Feature Teams

Backend, Frontend



Rider App  
Platform



Earners App  
Platform



Eats App  
Platform



**Developer Platform**

Part of Platform Engineering

4,500

Feature  
engineers

200

Developer Platform  
engineers

22:1

Support ratio

3+

Engineering sites

# Developer Experience

IDEs & Tools

Training & Documentation

App & Service Frameworks

Internal Libraries & Guardrails

Monorepo Tooling & Build System

6

Monorepos for Swift, Kotlin, Typescript, Go, Java, and Python

5<sub>k+</sub>

Microservices

3

Major mobile apps

+8

Net promoter score

6

Minor mobile apps

# Technical Debt

- Backlog of updates
- Fragmentation
- Test coverage

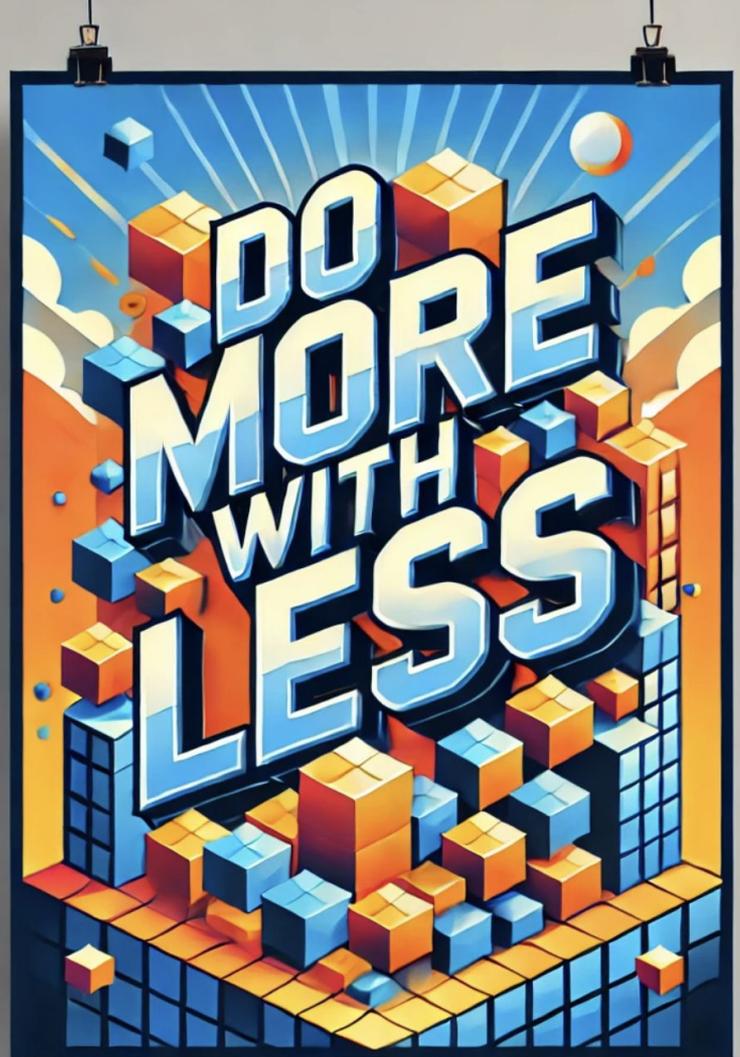
100<sub>million+</sub>

Lines of code across all of  
the monorepos



# Macro Trends

- Flat headcount
- Backfills not guaranteed



# Emergence of AI as Leverage

Can't scale people with the growth  
and maintenance needs of the  
codebase

Can we position Developer Platform  
to be AI-driven?

# Org High-level View

Created a  
**centralized AI  
DevEx team**  
that  
specializes in  
AI applied to  
the SDLC

Platform Engineering

Quality & Productivity Engineering

**Developer Platform**

Programming  
Systems

Code Infra

IDE

Mobile  
Platform

Backend  
Platform

Testing  
Automation

**AI Foundations & Developer Experience**

ML Infrastructure

# Timeline

## Applied-AI Developer Tools



### **Inaugural Hackdayz**

Oct 2022

### **The ChatGPT moment**

Nov 2022

### **Exploring generative AI**

Hackdayz Summer 2023

### **Building automation and becoming AI-driven**

Hackdayz Winter 2024

### **Focus on agentic systems**

Hackdayz Summer 2024

# Agentic Systems

Multi-step systems to interact with LLMs

Breaks down a problem space into manageable tasks

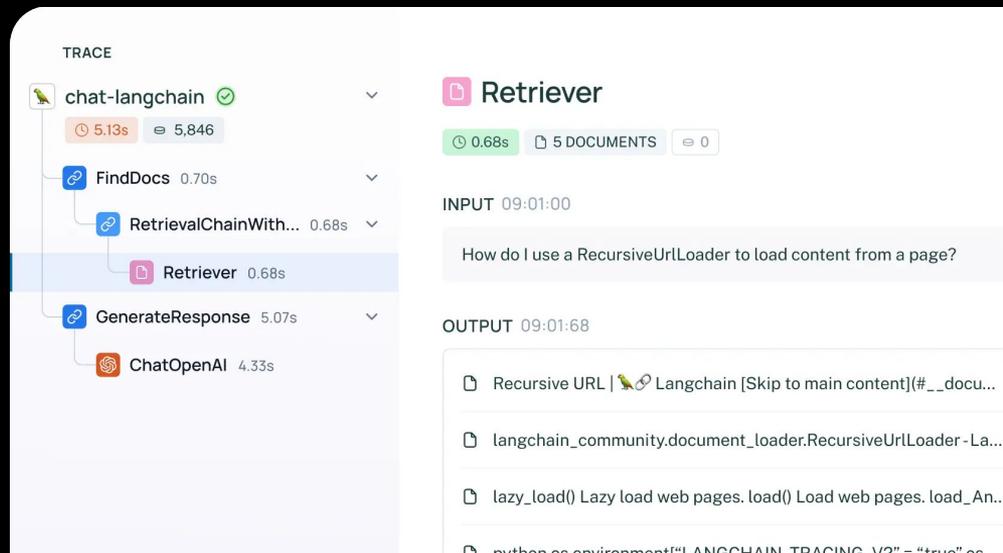
**You'll see some examples in our upcoming stories**



LangChain



LangGraph



The screenshot displays a 'TRACE' interface for a LangChain application. The main trace shows a sequence of steps: 'chat-langchain' (5.13s, 5,846 tokens), 'FindDocs' (0.70s), 'RetrievalChainWith...' (0.68s), 'GenerateResponse' (5.07s), and 'ChatOpenAI' (4.33s). The 'Retriever' step is highlighted in blue. A detailed view of the 'Retriever' step is shown on the right, indicating it took 0.68s and returned 5 documents. The input to the retriever is the question: 'How do I use a RecursiveUrlLoader to load content from a page?'. The output shows the first document snippet: 'Recursive URL | Langchain [Skip to main content] (#\_docu...'. Other document snippets include 'langchain\_community.document\_loader.RecursiveUrlLoader-La...' and 'lazy\_load() Lazy load web pages. load() Load web pages. load\_An...'.

# Applied-AI SDLC



1. Coding Assistants
2. Generating Tests
3. Java to Kotlin Migration



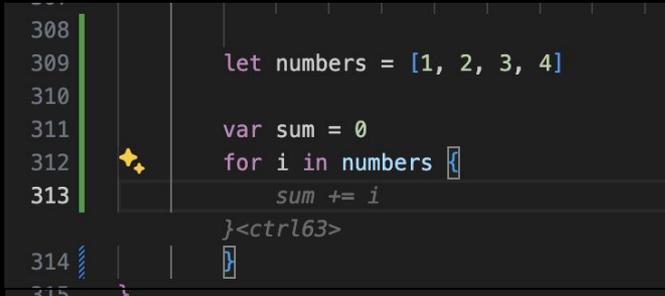
# Coding Assistants

# Code Assistants

## Basics

- Native Plugin with Language Intelligence
- Model Backend

## Contributes to **UX** and **Result Quality**



```
308  
309 let numbers = [1, 2, 3, 4]  
310  
311 var sum = 0  
312 for i in numbers {  
313     sum += i  
314     }<ctrl63>  
315
```

Broken Example Caused By IDE validation

## IDE Plugin

### Language Intelligence

Semantics, syntax, references & symbol navigation

### Verification

Deterministic fixes for generated suggestions.

### Extensibility

Adapt to development environments, workflows, & custom tools



## Model backend

Foundational model and context awareness

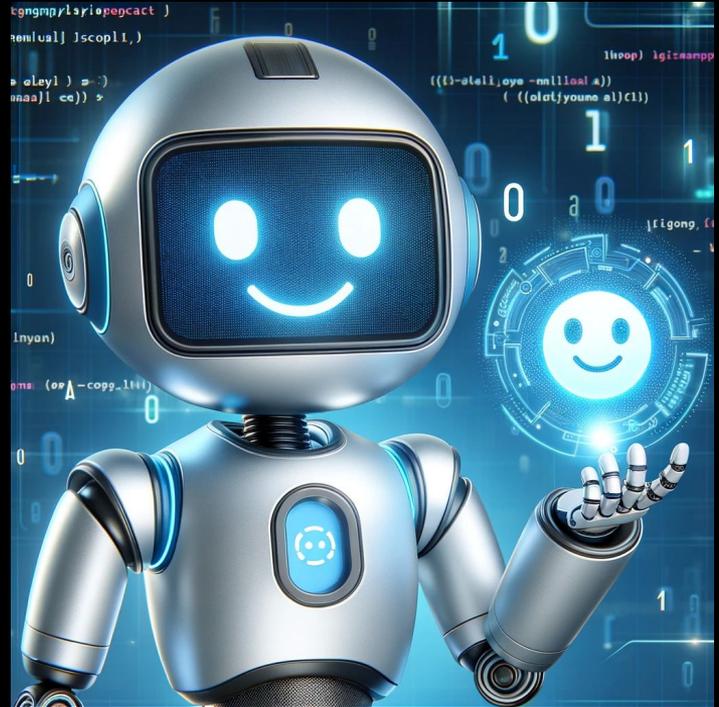
# Hypothesis

An Uber trained IDE  
assistant is needed

# Custom Code Assistants

## Requirements

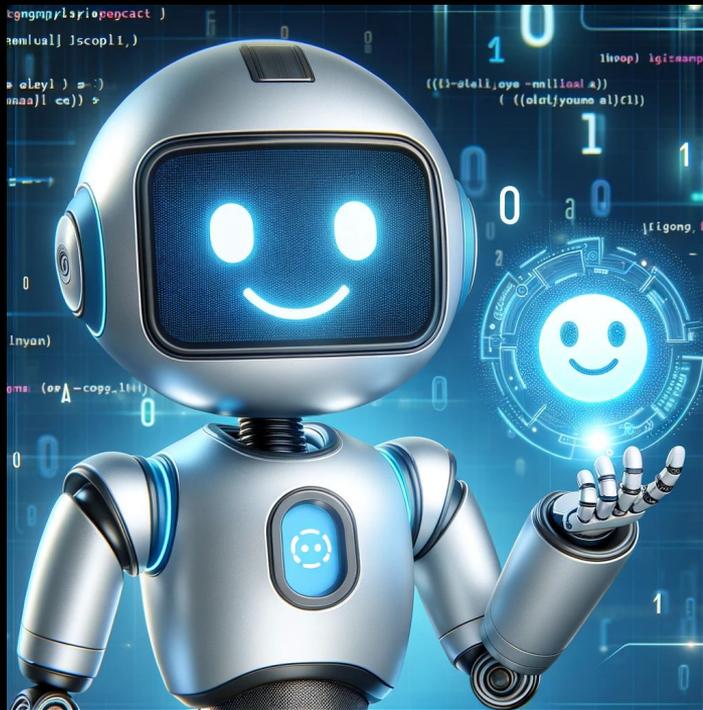
- Uber aware
- Fast
- Cost effective
- Per user analytics
- Workflow integrated



# Custom Code Assistants

## Buildout

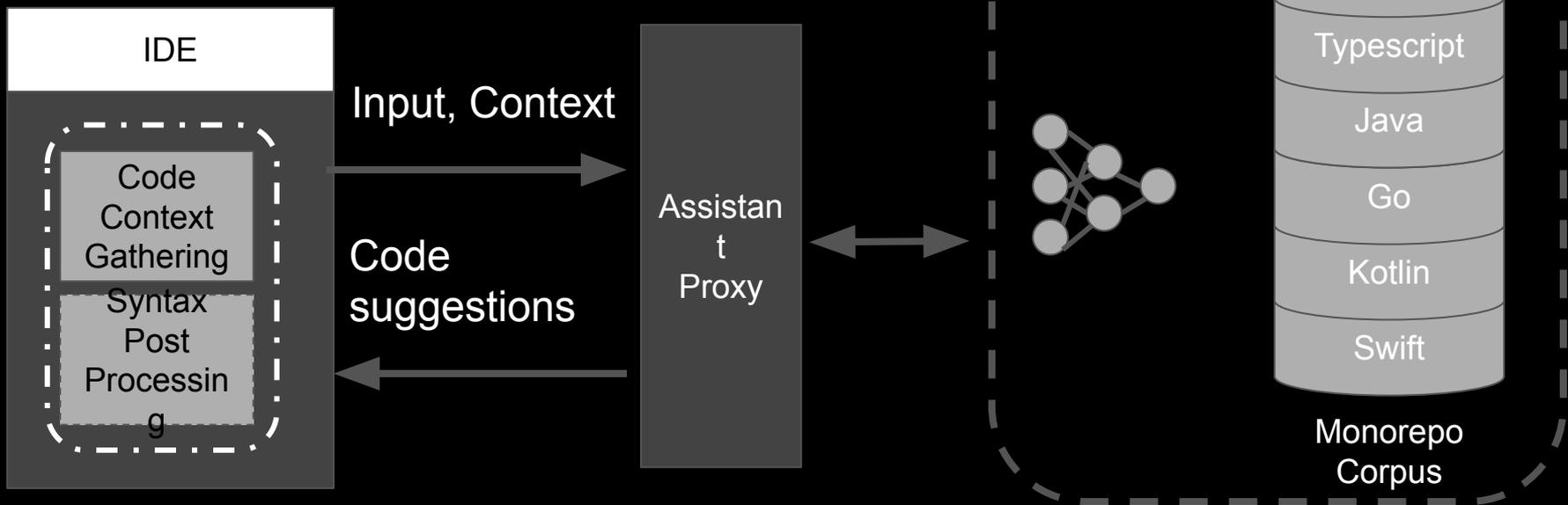
- MVP in Hackathon
- Evaluate LLMs
- Internal evangelism
- Wide variety of investments



# In-House Coding Assistant

## Goals:

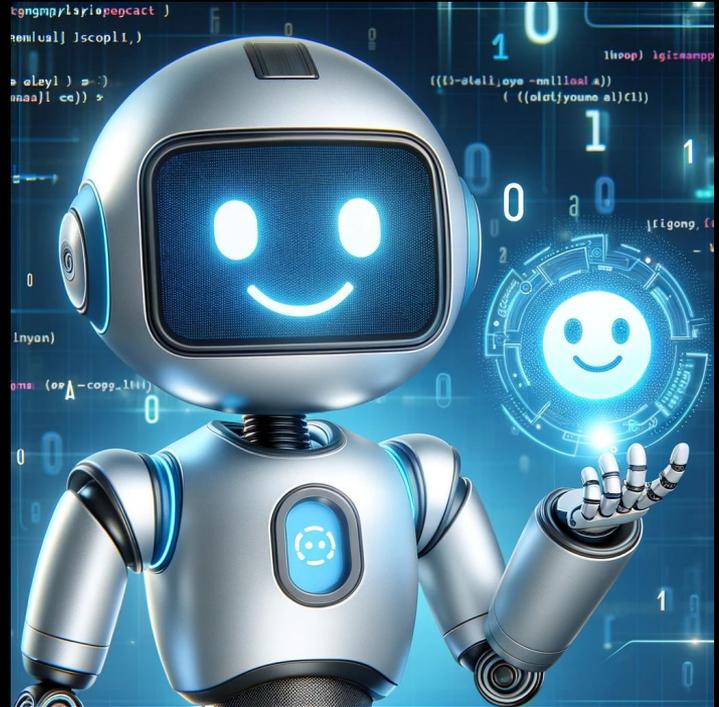
- Increase acceptance rate by +10%
- latency <1s for 100 tokens



# Code Assistants

## Downsides

- 6 months of work
- Underfunded
- Always playing catch up



# What we learned

- MVPs are easy, productionisation is hard
- Latency requirements vary per tool
- User Experience matters
- UI surface cannibalization is a risk
- Follow ecosystem principle
- Continuously evaluate landscape



Focused on GitHub Copilot  
adoption & evangelism

# Building on Industry Tools

# Reusable Components

Code Context Gathering

Summarize & rank code context to provide best input to use-cases:

Data Race  
Fixer

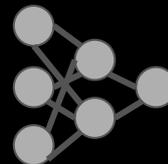
Lint  
Warning Fixer

Crash Fixer

Gather telemetry

Assistan  
t  
Proxy

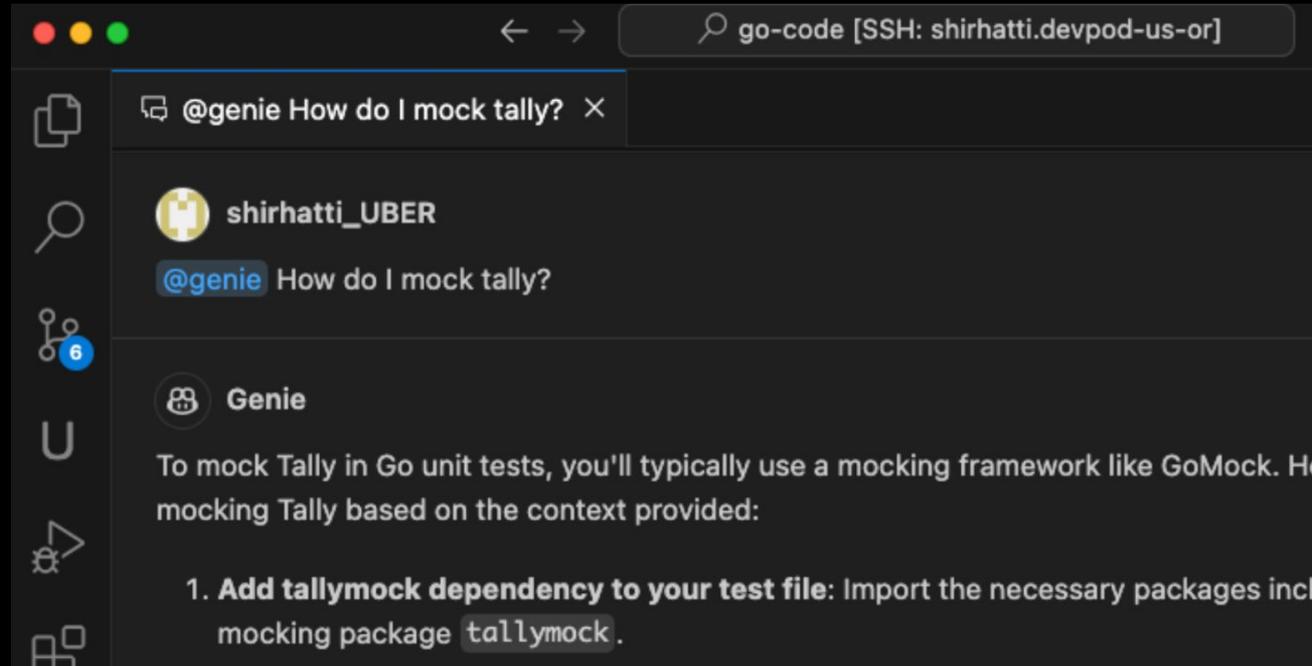
Fine-tuned LLM



Custom model with knowledge of internal libraries, custom frameworks, and company-specific best practices

# Extend with chat @participants

@genie for  
monorepo  
knowledge base  
queries



# Hands-on workshops

Running once per month to spread knowledge

Experience iterating with LLMs and probabilistic outputs



# Internal Evangelism Content

## Chat participants

@workspace  
@vscode

Extensible!

## Chat commands

/doc  
/explain  
/fix  
/tests  
...

## Chat context

#codebase  
#editor  
#file  
#selection  
....

A series of codelabs that teach these engineers about providing context, using chat participants, and commands to refactor code and generate tests

# Coding Assistants

## Future

- Platform Native
- Vendor Fine Tuning
- Extensibility Increases
- Open-source clients
- Multi Vendor Landscape
- Enterprise Requirements





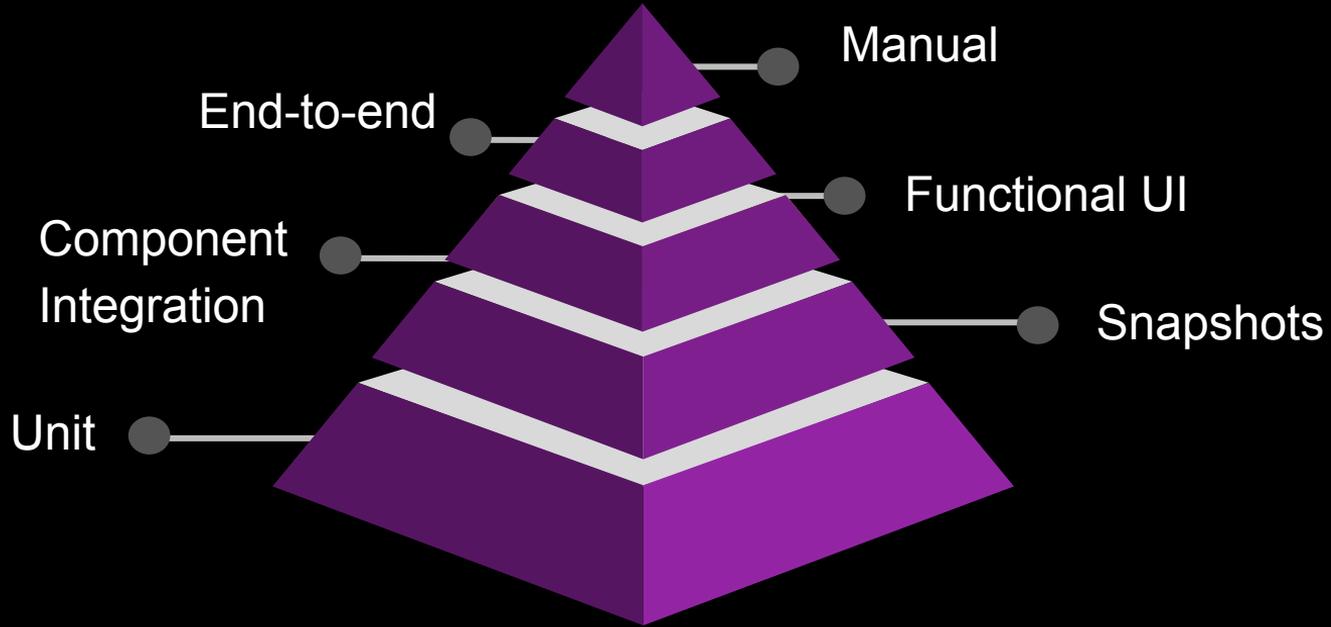
# Generating Tests

# Classic Testing Strategy

More interconnection,  
slower, fewer tests



More isolation,  
faster, more tests

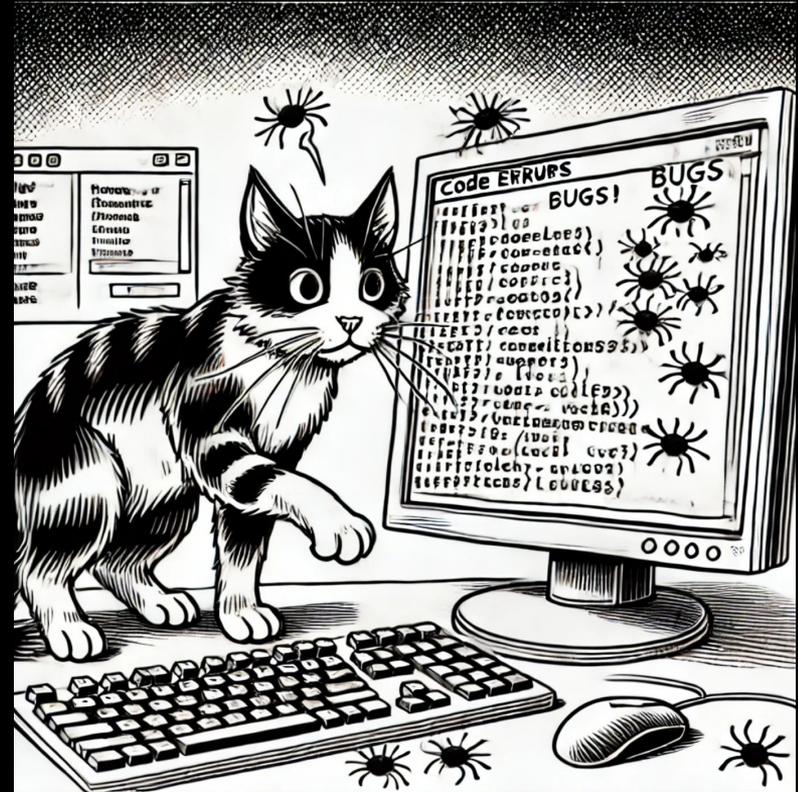


# Testing Challenges

Writing good tests can be hard

Maintaining many different types of tests is tedious

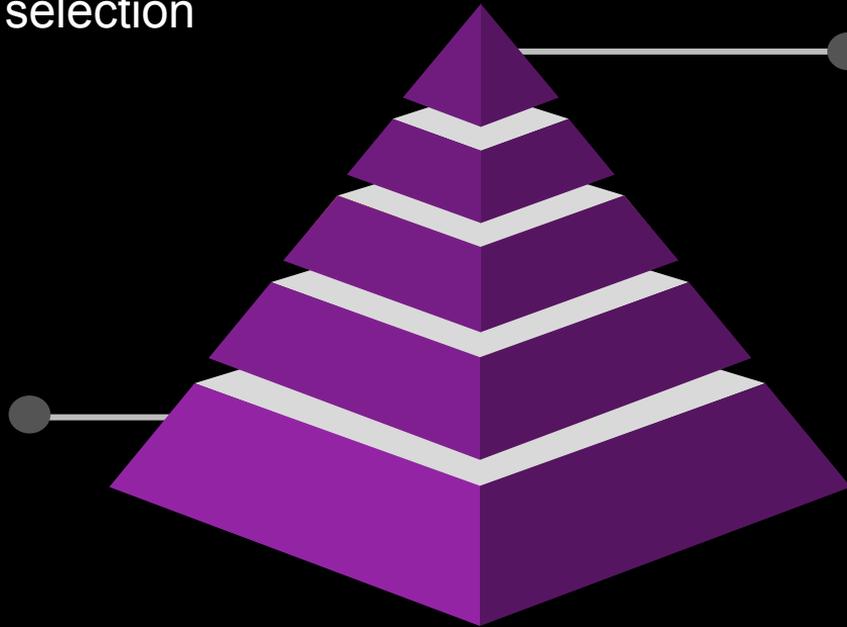
Permutations of languages, cities, experiments, platforms



# AI-Driven Testing Pyramid

- Analyze quality of tests
- Predictive test selection

AI-powered code  
generation of  
unit tests



End-to-end AI testing  
agents

# AutoCover Requirements

- Keep the developer-in-the loop
- Focus on regression tests
- Increase coverage



# AutoCover in action

Tests are streaming  
in...

The screenshot shows a chat window on the left and a terminal window on the right. The chat window has a header 'CHAT' and a '+' icon. It contains three messages:

- GitHub Copilot**:  
Welcome, @matas\_UBER, I'm your Copilot and I'm here to help you get things done faster. You can also [start an inline chat session](#).  
  
I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and [share feedback](#) so that we can learn and improve. Check out the [Copilot documentation](#) to learn more.
- matas\_UBER** (highlighted with a yellow box):  
`@autocover /start`
- Uber Auto Cover**:  
Okay! Generating tests now.

At the bottom of the chat is an input field with the text 'Ask Copilot' and icons for voice search, attachments, and a dropdown arrow.

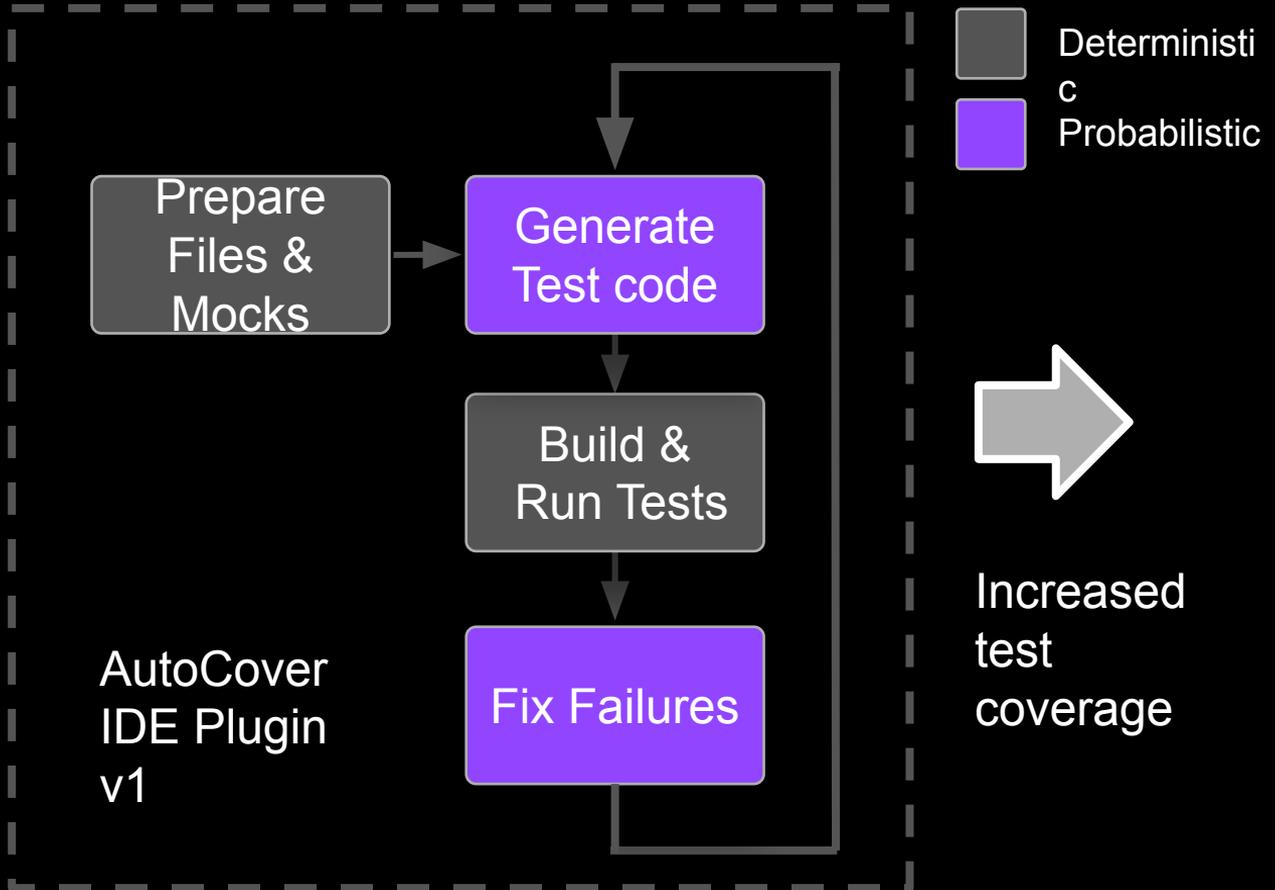
The terminal window on the right shows the following content:

```
src > code.uber.internal > crack > wallet > gateways > tasks > tasks_test.go
You, 6 days ago | 1 author (You)
1 package tasks
2
```

Break up the problem

Mimic human heuristics

Agentic design, built with LangGraph



Automating  
away all this  
tedious  
work...



This needs to be  
labeled “generated  
with AI!”

Is it writing  
“change  
detector” tests?

**Validation step:** Check assertions against intent

**Refactor step:** Adopt best practices like the table pattern

Prepare Files & Mocks

Generate Test code

Build & Run Tests

Fix Test Failures

Validate Test Quality

Refactor Table Test

AutoCover  
IDE Plugin  
v2

# What's next

## IDE/CLI for humans

- Table test refactor
- Validate test quality

## Headless mode

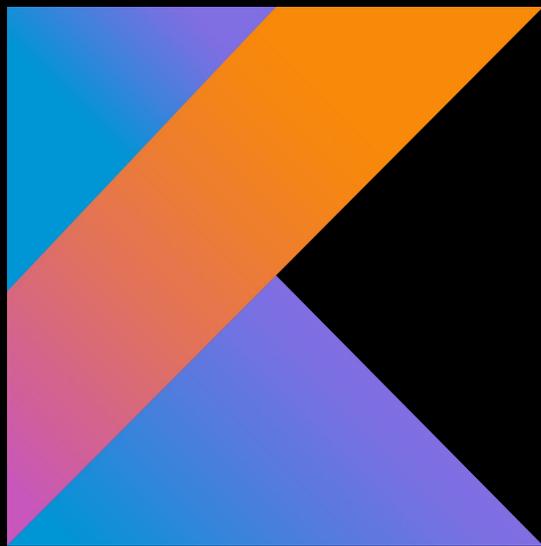
- Shift-right, runs on CI
- Improve quality of existing test code

## Mutation testing step

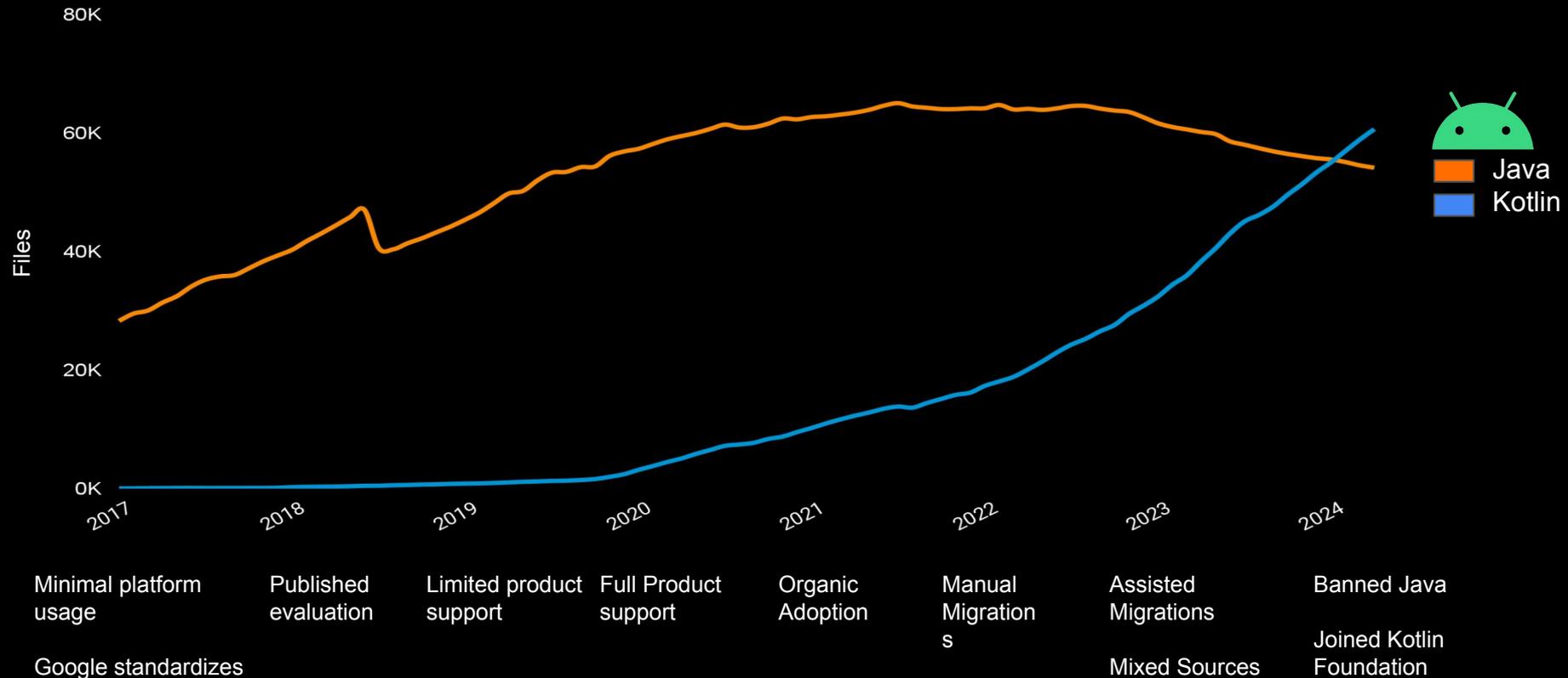
- Inject bugs (“mutants”) into the source code
- See if it finds bugs
- Generate mutants with AI



# Migrating To Kotlin



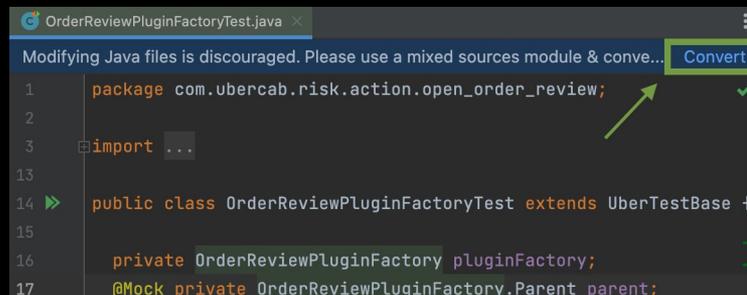
# Kotlin History at Uber



# Kotlin Migrations

## Decentralized

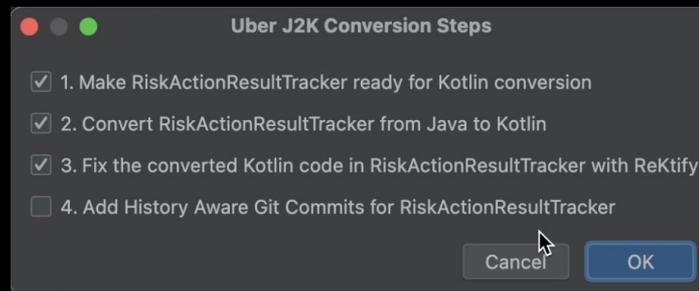
- Workflow incentives
- Industry standard
- Developer assisted



OrderReviewPluginFactoryTest.java x

Modifying Java files is discouraged. Please use a mixed sources module & converge... **Convert**

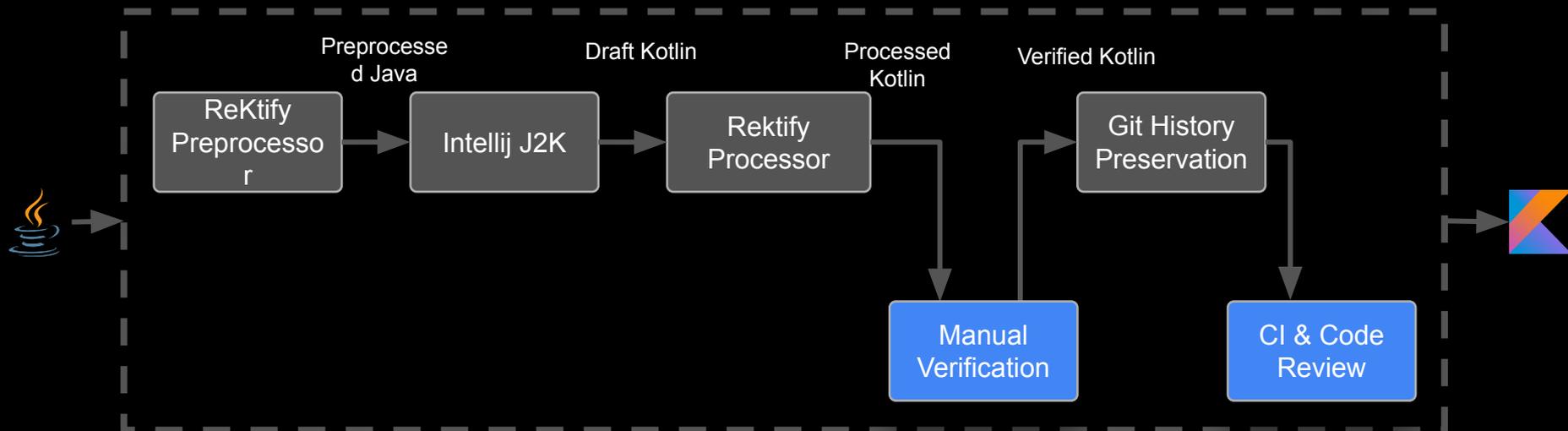
```
1 package com.ubercab.risk.action.open_order_review;
2
3 import ...
13
14 public class OrderReviewPluginFactoryTest extends UberTestBase {
15
16     private OrderReviewPluginFactory pluginFactory;
17     @Mock private OrderReviewPluginFactory.Parent parent;
```



# Kotlin Migrations Today

■ Tool

■ Human in the loop



# Rektify Processors

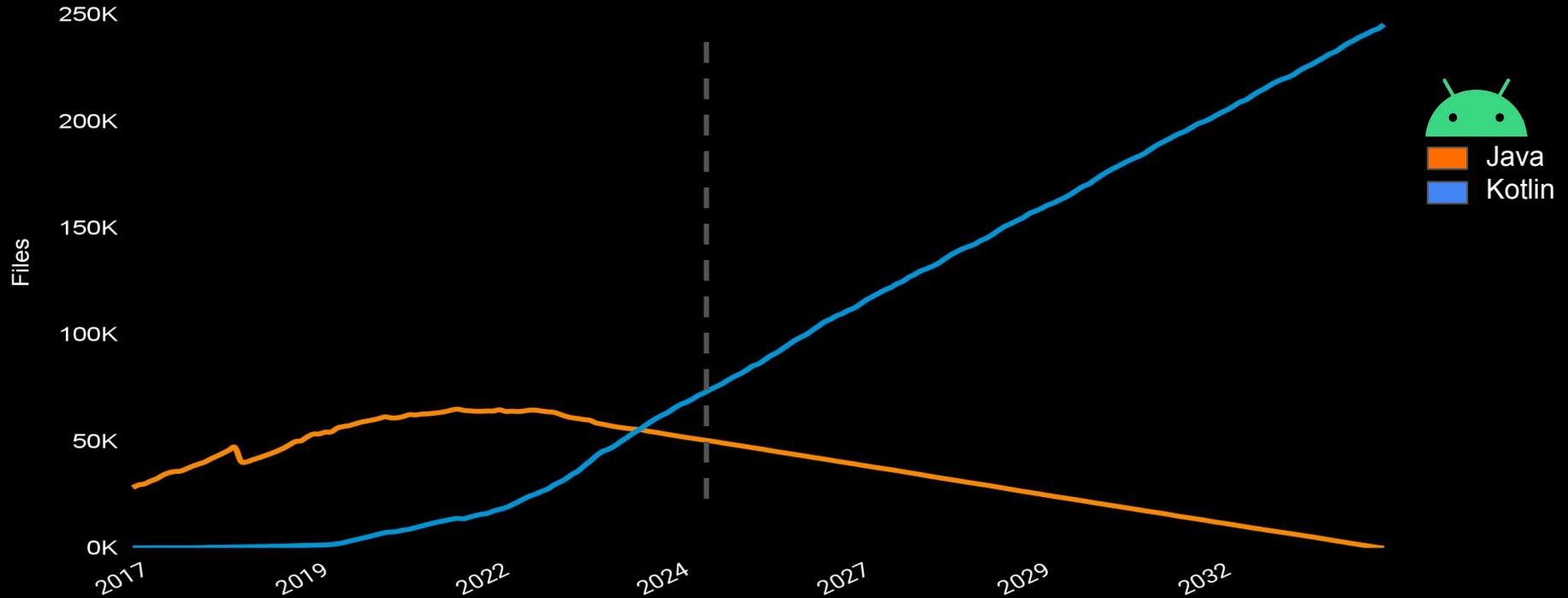
## Pre Processors

- Nullable Annotations

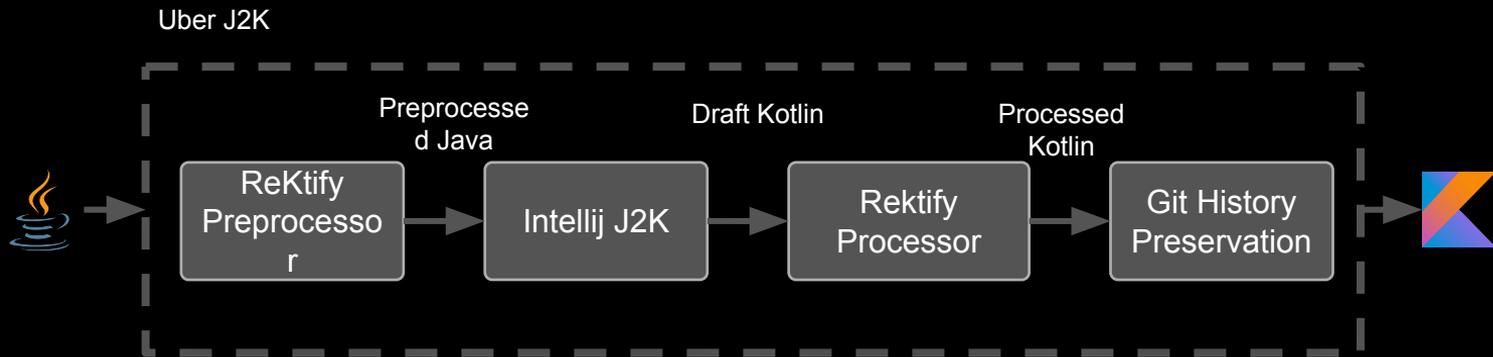
## Post Processors

- AndroidTextUtilsRule
- CaptorAnnotationRule
- GuavaStringUtilsRule
- LambdaExpressionRule
- MockAnnotationRule
- RemoveInitMocksRule
- AutoDisposeRule

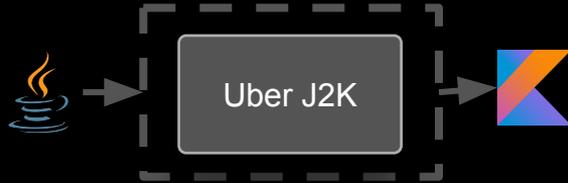
# Kotlin Migrations



# Automated Kotlin Migrations



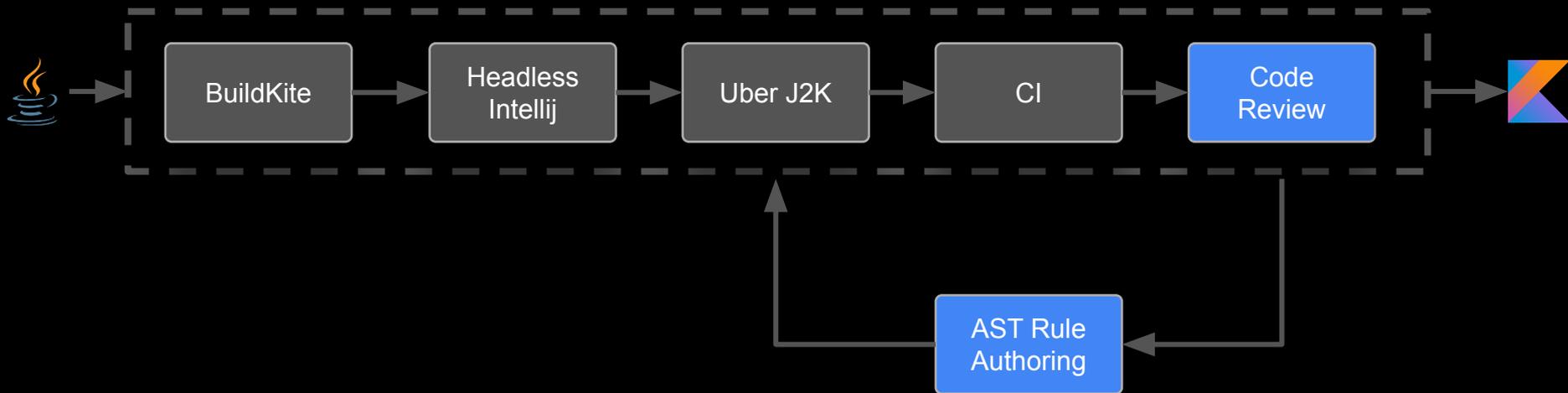
# Automated Kotlin Migrations



# Automated Kotlin Migrations



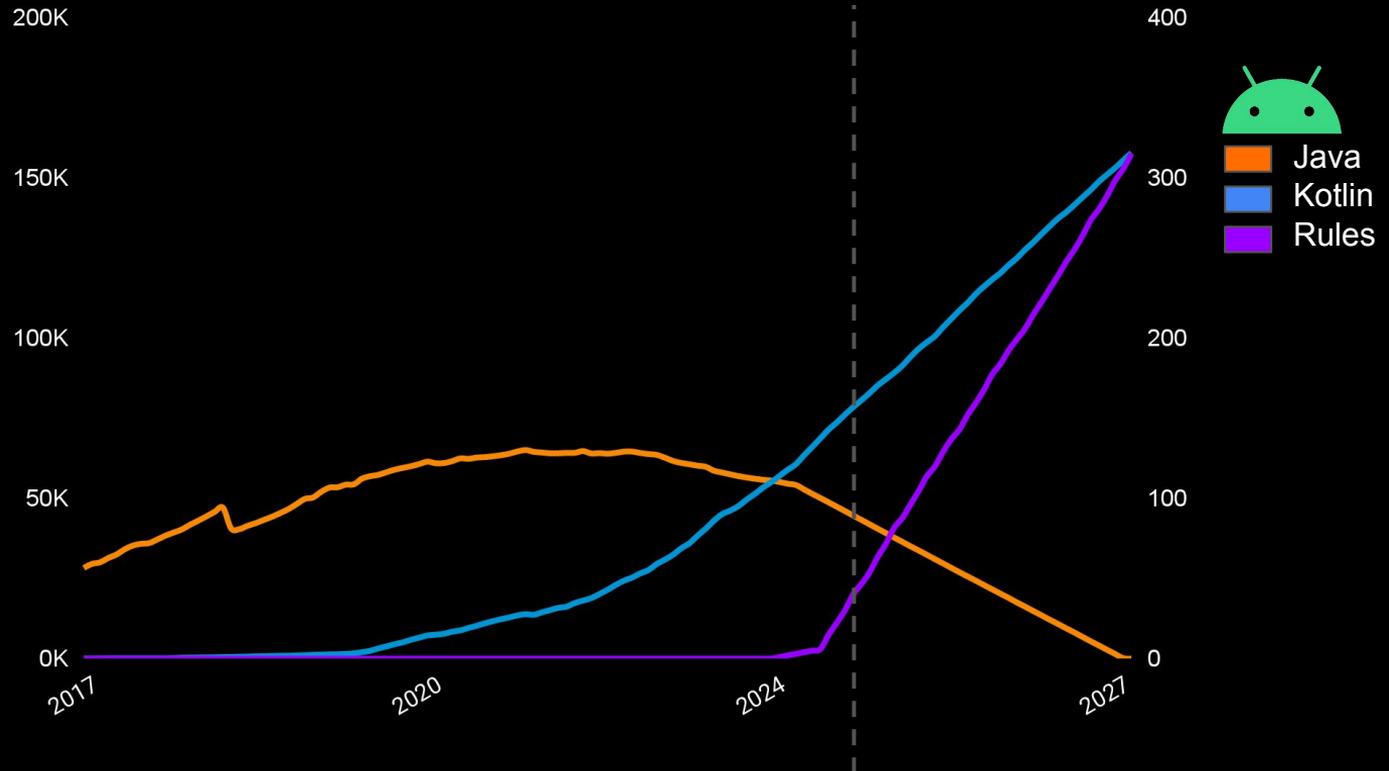
# Automated Kotlin Migrations



# Kotlin Migrations

## Centralized

- Industry group
- ~3 Years



Can AI go faster?

# LLM Kotlin Migration

## Positives

- Flexible use cases
- Fast to deploy

## Negatives

- Hallucinations possible
- High risk failures
- Humans are fallible

# Combining AST + LLMs

# AST + LLM

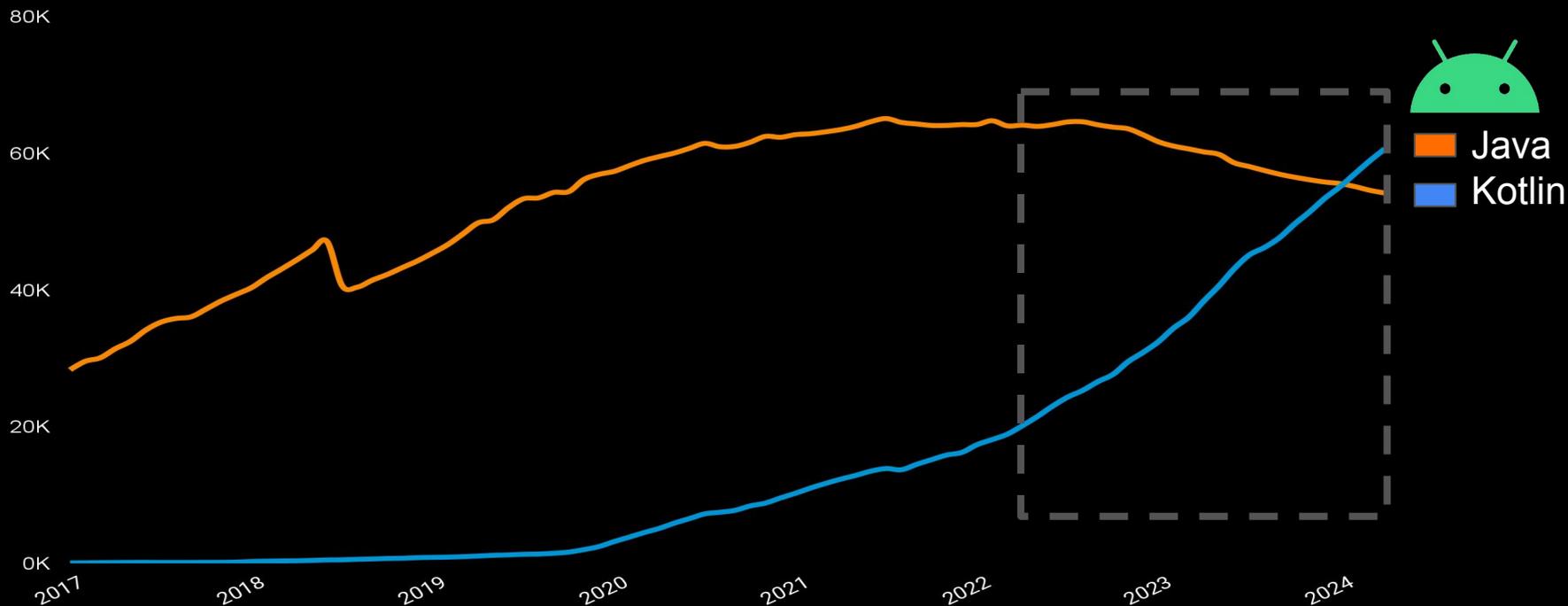
## Positives

- Prior Art
- Deterministic
- Faster than human authored Rules

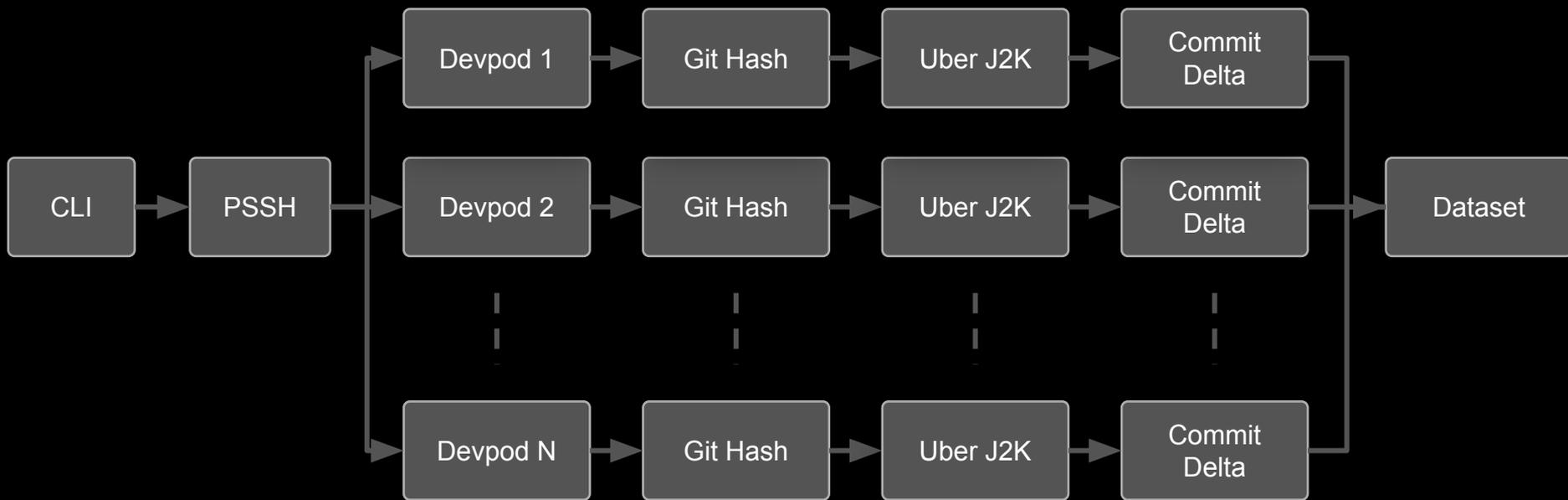
## Negatives

- Slower than LLM only

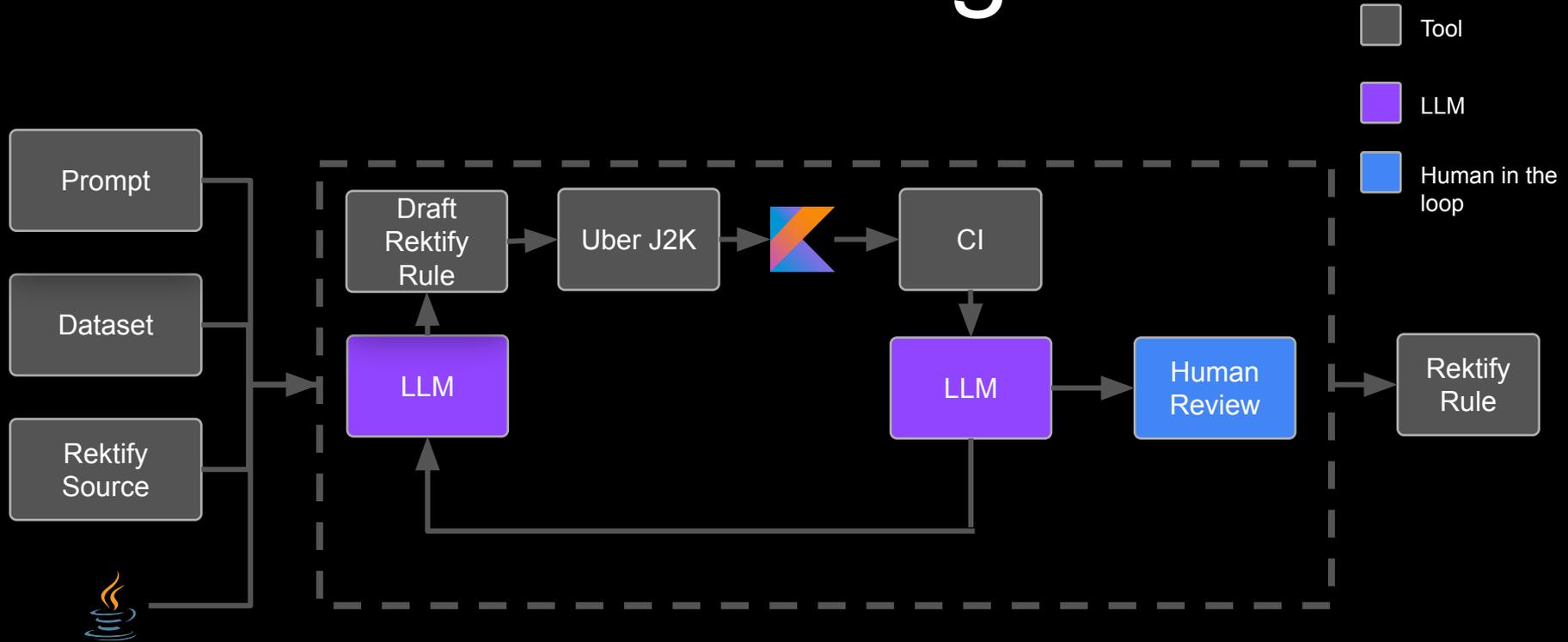
# Dataset



# Dataset



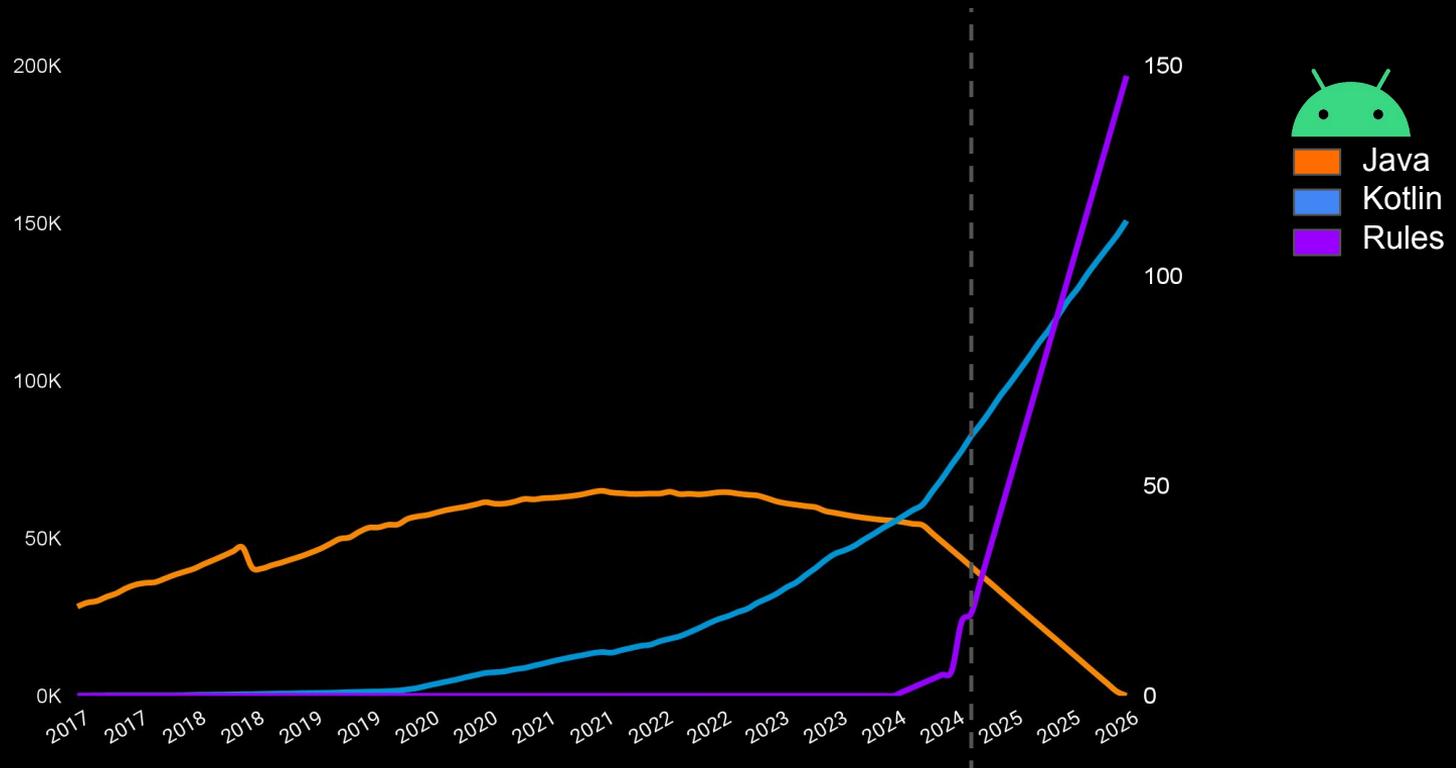
# LLM+AST Kotlin Migrations



# Kotlin Migrations

## LLM+AST

- 50% Faster
- 18 Months



# Finishing the Migration

## Challenges

- Rollout risk
- Begin in low risk areas
- Categorization of rules
- Noise fatigue

## Questions

- LLM fallbacks
- Batch size
- Speed of innovation

Wrap up

# Measuring AI impact

~10

%

PR velocity increase

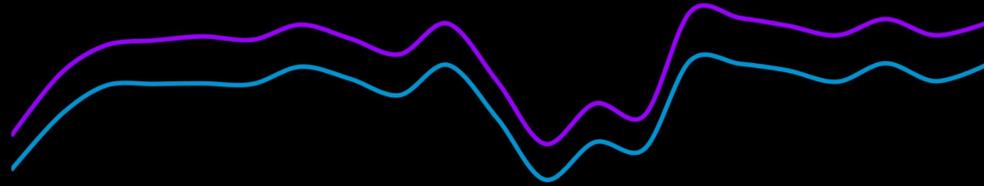
~30%

Acceptance Rate

~60%

Adoption

— Non-Copilot Users  
— Copilot Users



Jan, 2023

Jul, 2023

Jan, 2024

Jul, 2024

# Measuring AI impact

## Challenges

- Fragmentation
- Organizational Cost

## Risks

- Side Effects
- Wrong Investments
- Missed Investments

# Measurement Philosophy

Lead with qualitative

Normalize quantitative impact on developer hours saved to prioritize bets

63%

Developers report **significant** increase in productivity

1000+ years

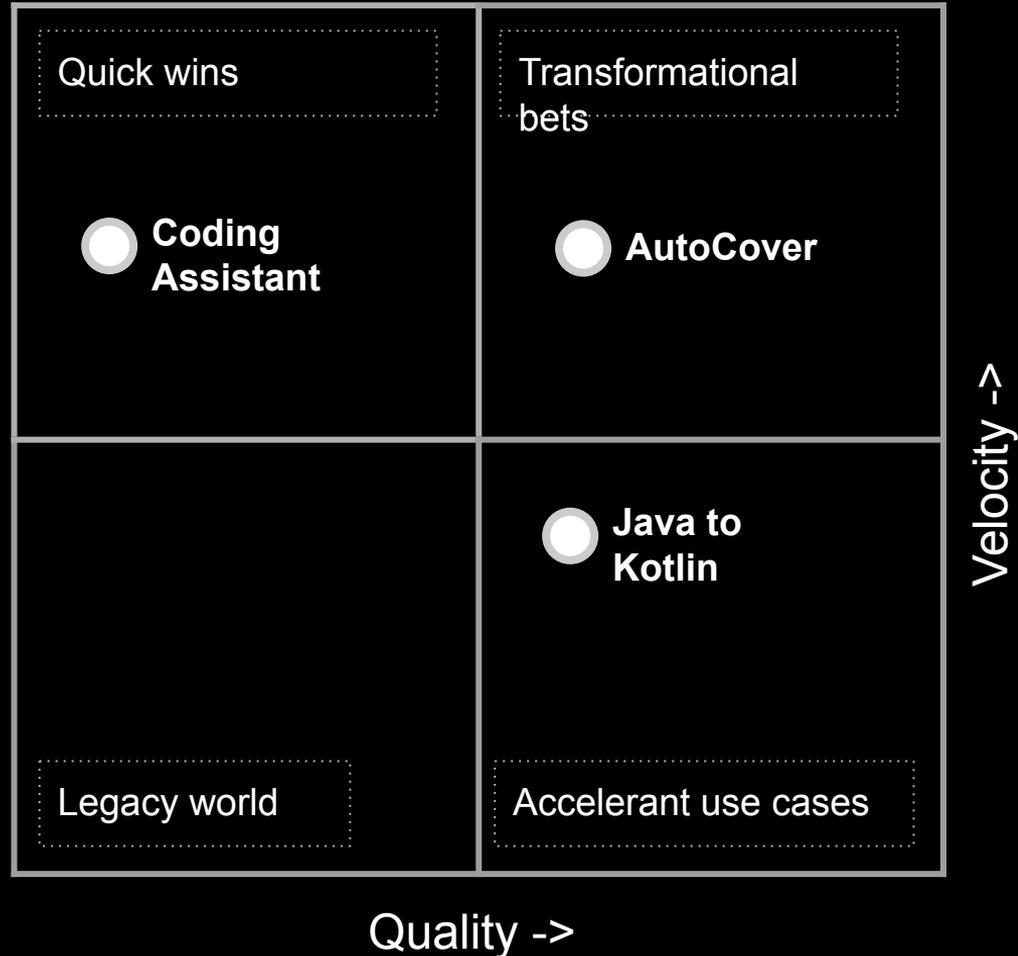
Potential impact of automating away of technical debt

# Opportunities

Quality and velocity

Manage expectations  
and hype

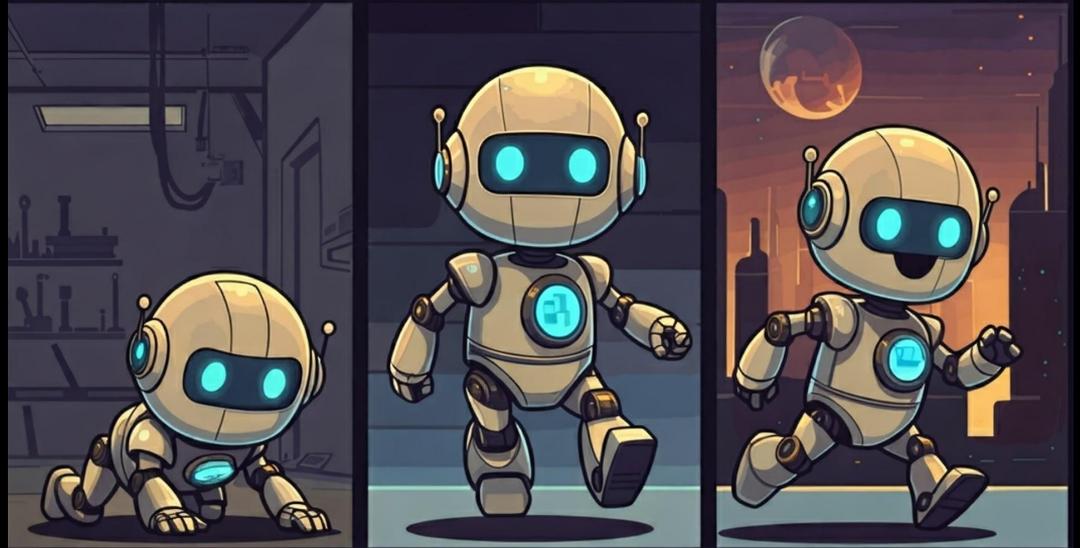
Combine deterministic  
approaches with new  
probabilistic capabilities



# Crawl, walk, run

Find the sweet spot  
of what's possible  
now and will be  
possible soon

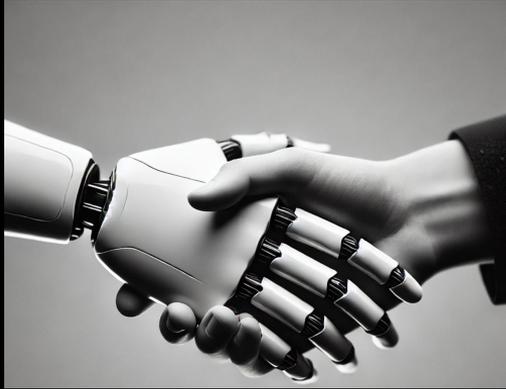
But be ready for  
what's coming



# Demand for good software is

near AI<sup>∞</sup>

- Reduce toil for migrations
- Increase test coverage
- Give humans more options
- Help humans think about problems



Humans

- More time building
- Focus time on the craft of software engineering
- Break down complex problems
- Define architecture
- Set best practices

# Questions?



**Adam Huda**  
Sr. Eng Manager  
AI Foundations  
& Developer Experience



**Ty Smith**  
Principal Eng  
Developer Platform

