LLM Agents for Program Repair and Project Setup

Michael Pradel (University of Stuttgart)

Joint work with Islem Bouzenia and Prem Devanbu

LLMs in Software Engineering

Most work on applying LLMs in SE:

Fixed set of relevant information

↓

Hard-coded prompt

→

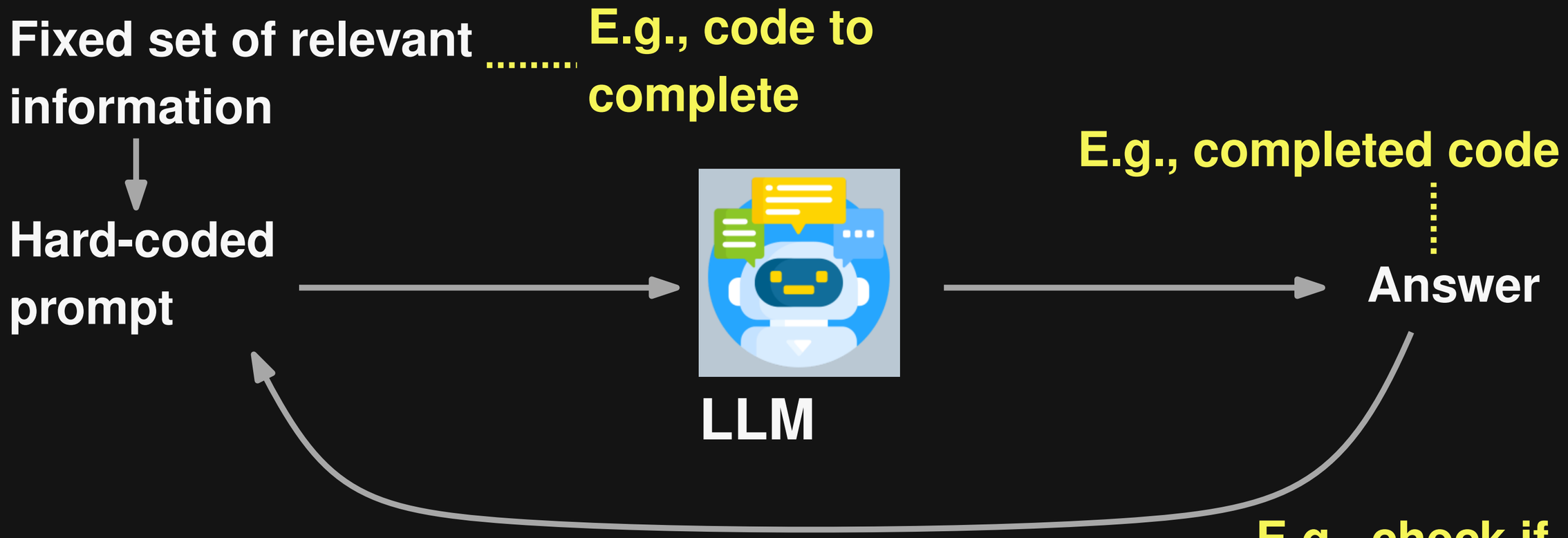
LLM

→ Answer

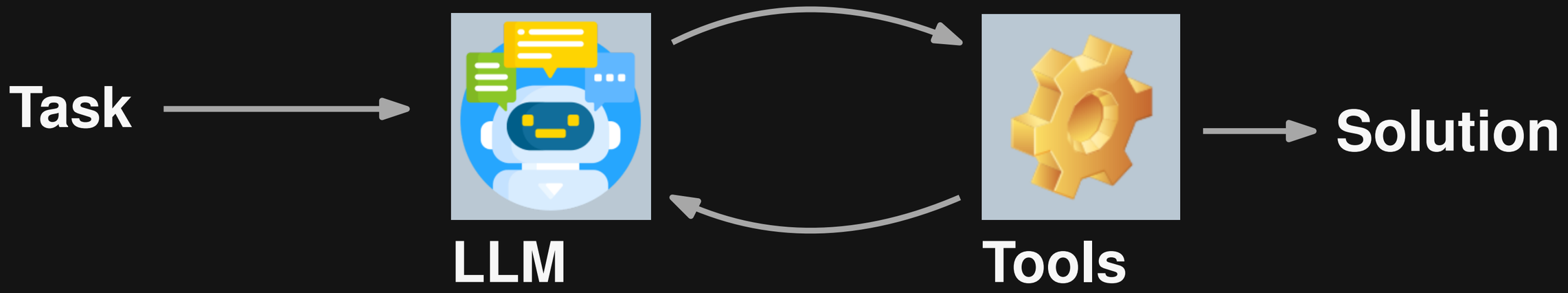
Optionally: Repeat until answer validated

LLMs in Software Engineering

Most work on applying LLMs in SE:



LLM Agents



- **Autonomous interaction** with environment
- **Mimics human** developers

This Talk

1) **RepairAgent**

- **Task: Automated program repair**

2) **ExecutionAgent**

- **Task: Build a project and execute its tests**

Example: RepairAgent

LLM agent

You are an AI assistant specialized in fixing bugs in Java code. Your objective is to autonomously understand and fix the following bug:
[Details of failing test case]

Use the following tools:
[Details on available tools]

Tools

Example: RepairAgent

LLM agent

Based on the stack trace, I should analyze the code in “ControlFlowAnalysis.java”.

Let’s call the ”read code” tool!

Tools

Example: RepairAgent

LLM agent

```
...
if (cfa != null) {
  for (Node finallyNode:
        cfa.finallyMap.get(parent)) {
    cfa.createEdge(fromNode,
                   Branch.UNCOND,
                   finallyNode);
  }
}
...
```

Tools

Example: RepairAgent

LLM agent

I need to see the failing test case “testDeepNestedFinally” to understand the assertions and inputs that lead to the failure.

Let’s call the “extract test code” tool!

Tools

Example: RepairAgent

LLM agent

```
...
String src = ...;
ControlFlowGraph<Node> cfg = createCfg(src);
assertCrossEdge(cfg, Token.THROW,
                     Token.BLOCK, Branch.ON_EX);
assertCrossEdge(cfg, Token.VAR,
                     Token.BLOCK, Branch.UNCOND);
...
```

Tools

Example: RepairAgent

LLM agent

Based on the test case and the code, I hypothesize that the use of ’Branch.UNCOND’ in ’finally’ blocks is incorrect

Let’s work under this hypothesis!

Tools

Example: RepairAgent

LLM agent

I should look at calls similar to the potentially buggy call to “createEdge”. This will help me understand ...

Let’s use the “extract similar calls” tool!

Tools

Example: RepairAgent

LLM agent

Search results:

```
createEdge(node,
                Branch.ON_TRUE,
                computeFallThrough(thenBlock));

createEdge(cfgNode,
                Branch.ON_EX,
                catchBlock);

...
```

Tools

Example: RepairAgent

LLM agent

Another call uses ’Branch.ON_EX’, which ...

Let’s apply and test this fix!

```
   cfa.createEdge(fromNode,
-                 Branch.UNCOND,
+                 Branch.ON_EX,
                  finallyNode);
```

Tools

Example: RepairAgent

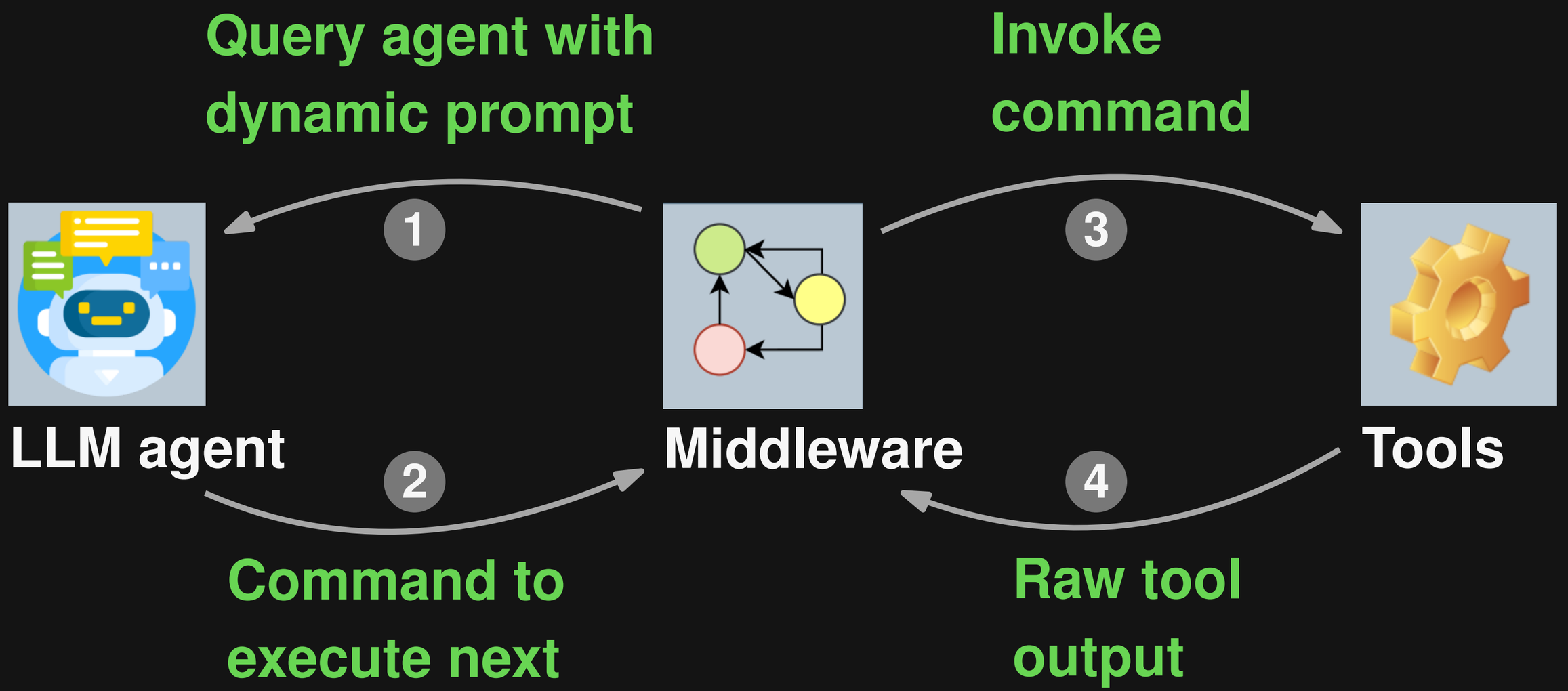
LLM agent

With this fix, all tests pass.

We are done!

Tools

Overview of RepairAgent

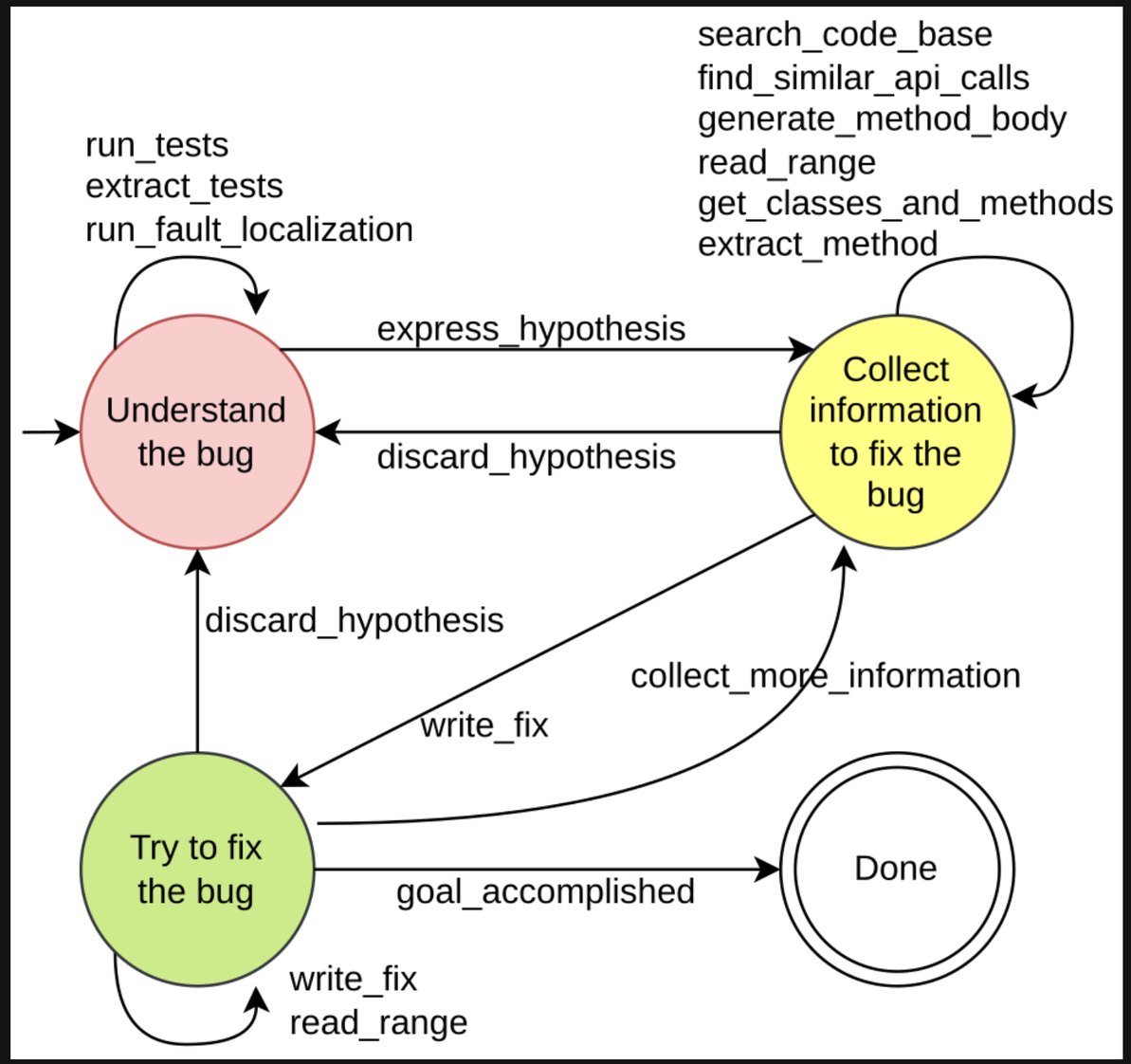


Tools

Category	Tools
Read and extract code	Read range of lines
	Get classes and methods
Search and generate code	Search code base by keyword
	Predict method body via LLM
Testing and patching	Run test suite
	Apply a patch
Control	Express or discard hypothesis
	Declare success

See paper for complete list of 14 tools.

Guidance via Finite State Machine



Format to Specify Patches

```
[ { "file_path": "jfree/data/time/Week.java",
    "insertions": [
        { "line_number": 175,
          "new_lines": [
            "// ...new lines to insert...\n",
            "// ...more new lines...\n"] }
    ],
    "deletions": [179, 183],
    "modifications": [
      { "line_number": 179,
        "modified_line": "if (dataset == null) {\n" }
    ]
  },
  { "file_path": "org/jfree/data/time/Day.java",
    "insertions": [],
    "deletions": [307],
    "modifications": []
  } ]
```

Format to Specify Patches

```
[ { "file_path": "jfree/data/time/Week.java",
    "insertions": [
        { "line_number": 175,
          "new_lines": [
            "// ...new lines to insert...\n",
            "// ...more new lines...\n"] }
    ],
    "deletions": [179, 183],
    "modifications": [
      { "line_number": 179,
        "modified_line": "if (dataset == null) {\n" }
    ]
  },
  { "file_path": "org/jfree/data/time/Day.java",
    "insertions": [],
    "deletions": [307],
    "modifications": []
  } ]
```

Support for multi-file edits

Format to Specify Patches

```
[ { "file_path": "jfree/data/time/Week.java",
    "insertions": [
        { "line_number": 175,
          "new_lines": [
            "// ...new lines to insert...\n",
            "// ...more new lines...\n"] }
    ],
    "deletions": [179, 183],
    "modifications": [
      { "line_number": 179,
        "modified_line": "if (dataset == null) {\n" }
    ]
  },
  { "file_path": "org/jfree/data/time/Day.java",
    "insertions": [],
    "deletions": [307],
    "modifications": []
  } ]
```

Support for multi-location edits within a file

Budgeting

- **Without budgeting: Endless, untargeted interactions**
- **Instead: Set a fixed budget of cycles**
  - Remind agent at every turn
  - Default: 40 cycles

Evaluation

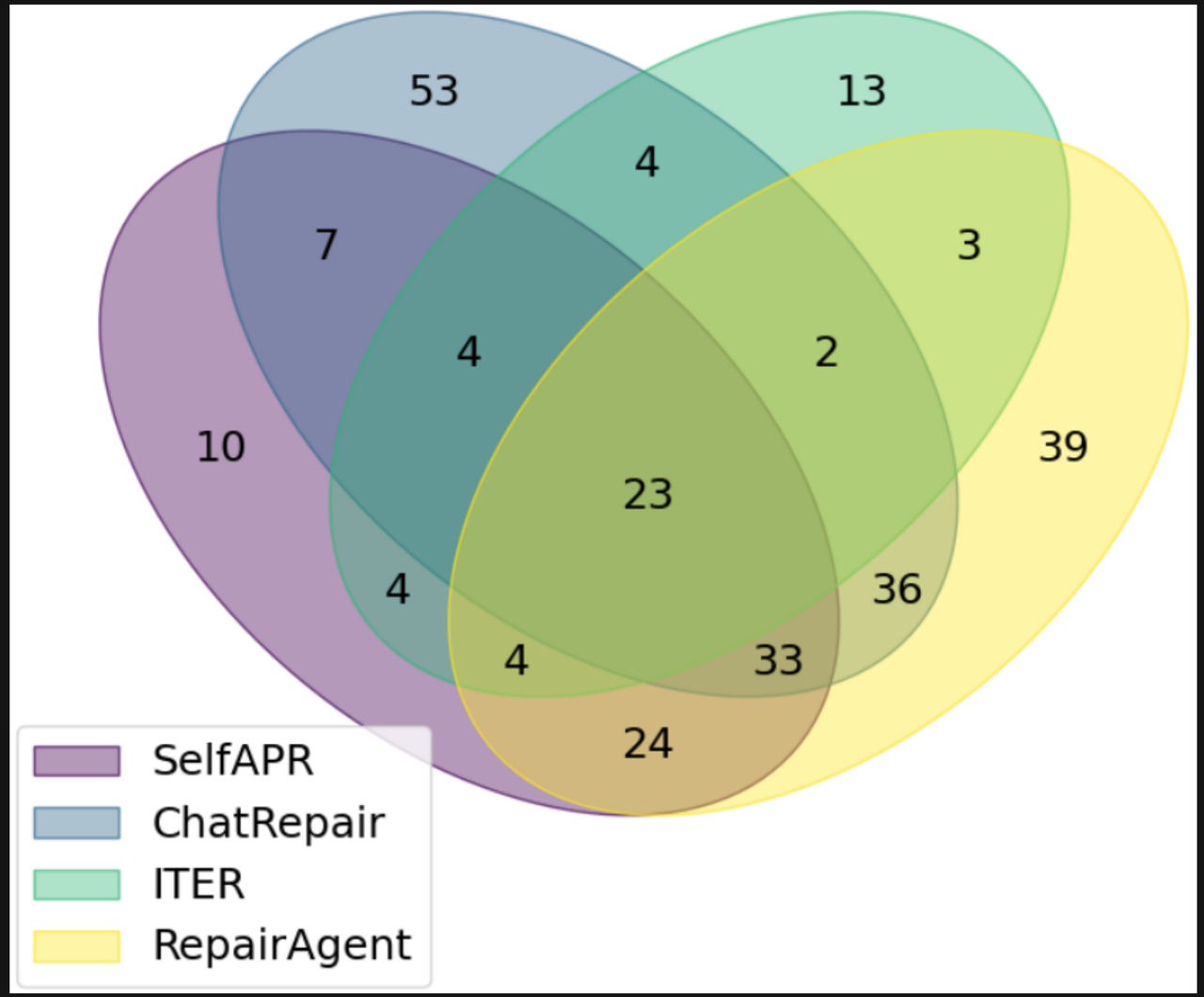
- **OpenAI's GPT-3.5-0125**
- **All 835 bugs from Defects4J v1.2 and v2**
  - Including multi-line, multi-file bugs
- **Measures of success**
  - Plausible fixes
  - Correct fixes
  - Cost per bug

Effectiveness

Correct bug fixes:

Bug type	RepairAgent	ChatRepair	ITER	SelfAPR
Single-line	110	133	36	83
Multi-line	46	29	14	24
Multi-file	3	0	4	3
Total	164	162	57	110

Effectiveness



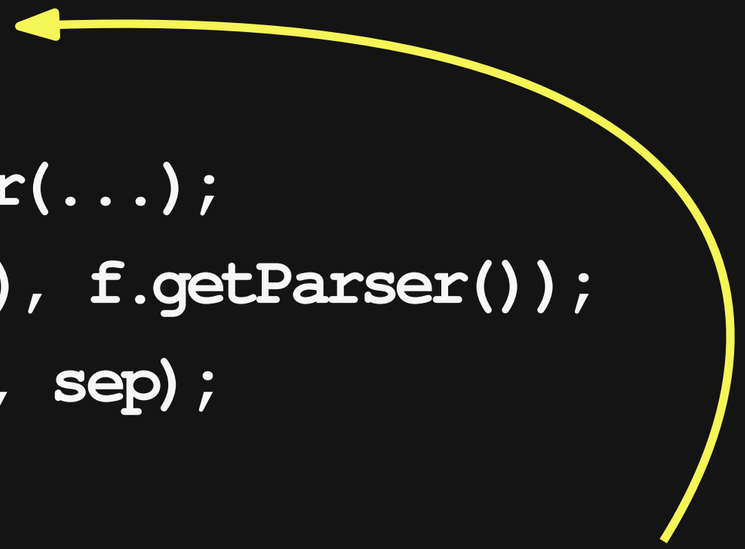
Examples

```
  if (cfa != null) {
    for (Node finallyNode : cfa.finallyMap.get(parent)) {
-     cfa.createEdge(fromNode, Branch.UNCOND, finallyNode);
+     cfa.createEdge(fromNode, Branch.ON_EX, finallyNode);
    }
  }
```

Found this field by searching the code base

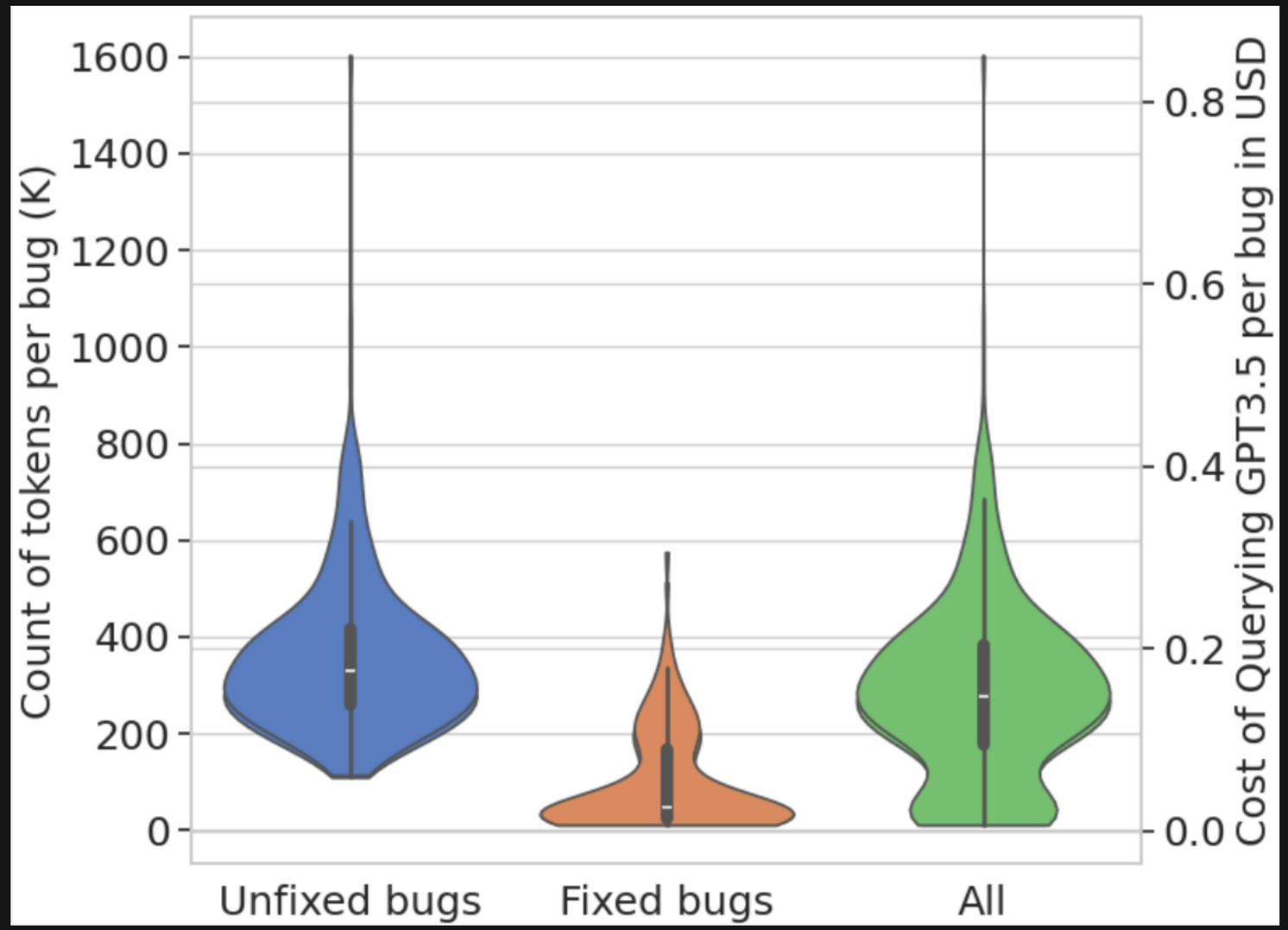
Examples

```
  Separator sep = (Separator) elementPairs.get(0);
+ if (sep.iAfterParser == null &&
+     sep.iAfterPrinter == null) {
    PeriodFormatter f = toFormatter(...);
    sep = sep.finish(f.getPrinter(), f.getParser());
    return new PeriodFormatter(sep, sep);
+ }
```



Found condition via LLM-based code completion

Costs



Avg. per bug:
270k tokens,
USD 0.14

This Talk

1) **RepairAgent**

- Task: Automated program repair

2) **ExecutionAgent**

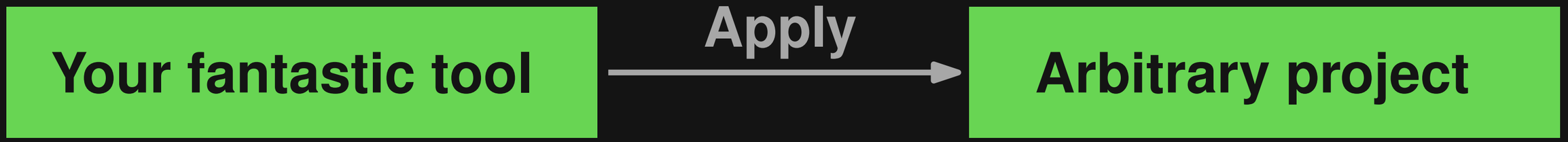
- Task: Build a project and execute its tests

This Talk

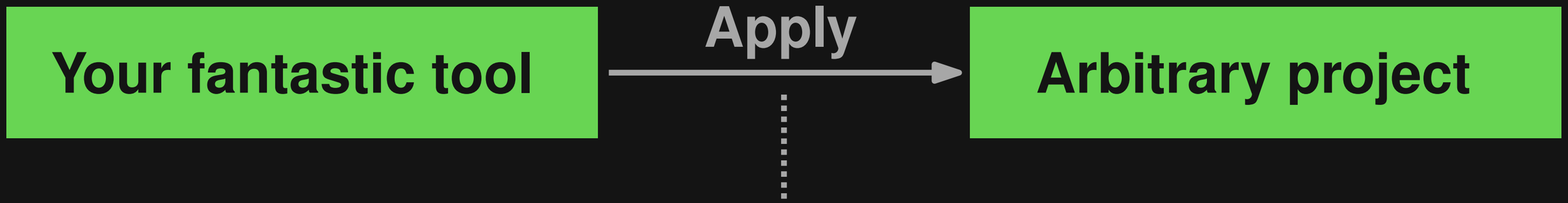
1) **RepairAgent**
   - Task: Automated program repair

2) **ExecutionAgent**
   - Task: Build a project and execute its tests

Motivation



Motivation



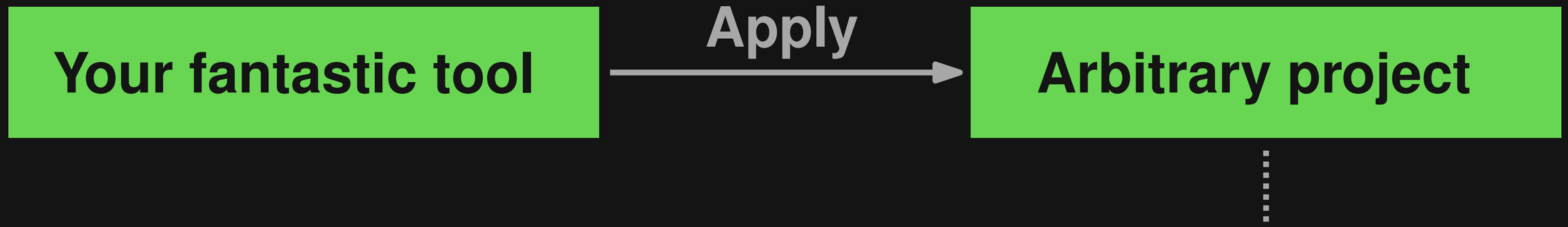
Prerequisite for many automated SE tools:
Project is set up and tests are ready-to-run

E.g., AI software engineer *

- Validate code edits via testing

* *AI Software Engineer: Programming with Trust*
Abhik Roychoudhury, Corina Pasareanu, Michael Pradel, Baishakhi Ray

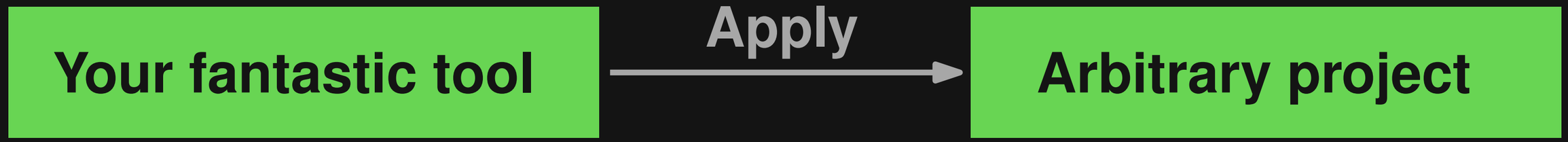
Motivation



Setting up an arbitrary project is hard

- Many languages, build tools, and test tools
- Complex dependencies
- Informal, inconsistent, and missing documentation

Motivation



Setting up an arbitrary project is hard

- Many languages, build tools, and test tools
- Complex dependencies
- Informal, inconsistent, and missing documentation

E.g., automated scripts of GitBugs-Java:
66,042 repositories $\rightarrow$ 228 executable test suites

Goal

URL of git repository

Scripts to

- set up project in isolated environment
- execute test suite of project

Goal

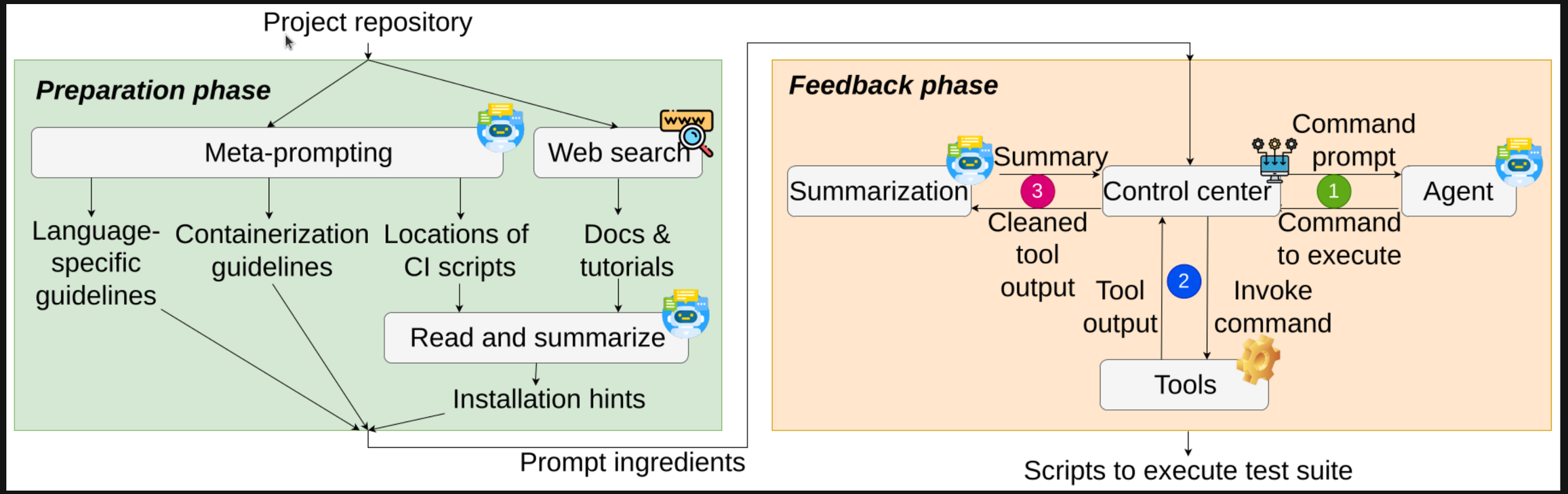
URL of git repository

Scripts to

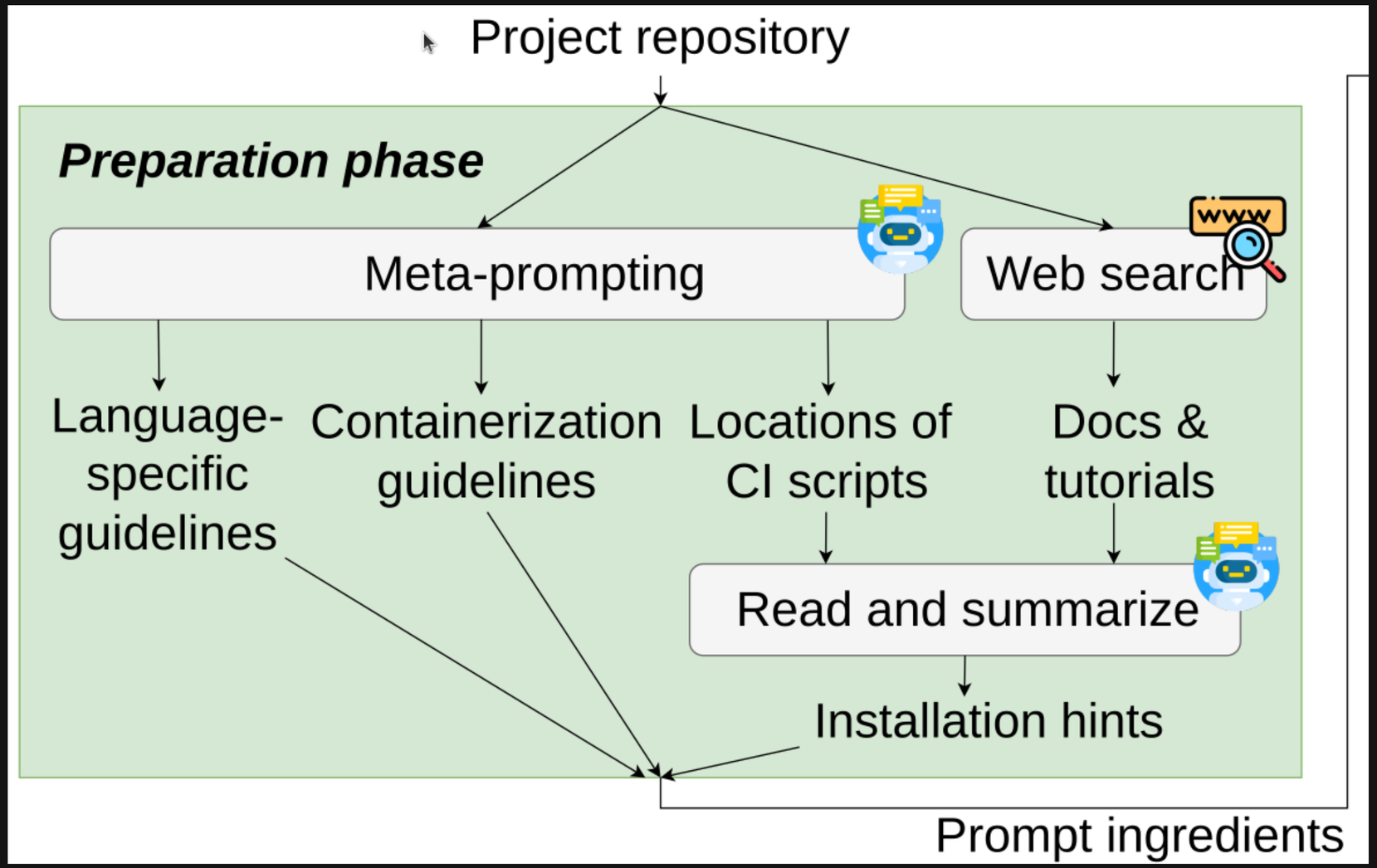
- set up project in isolated environment
- execute test suite of project

- **Fully automated**
- **Technology-agnostic**
- **LLM agent**

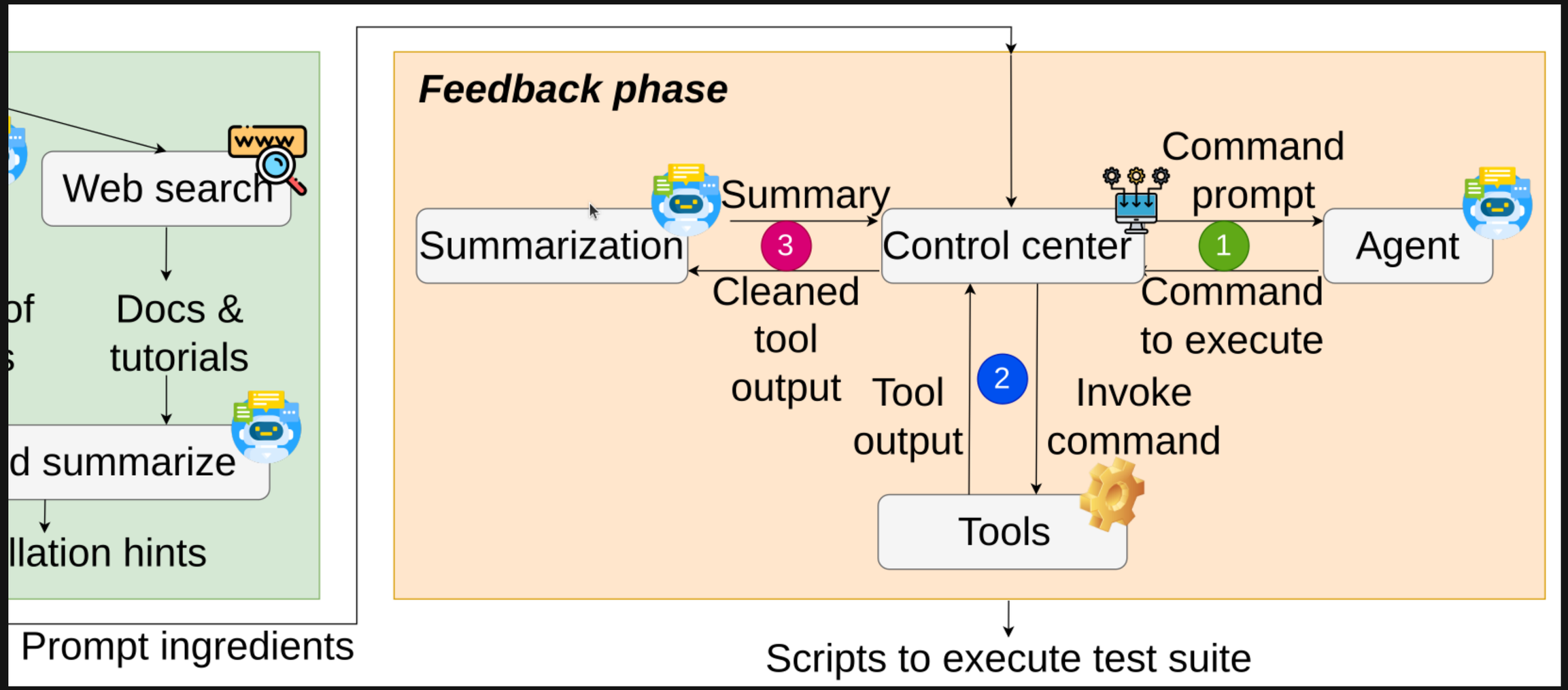
Overview of ExecutionAgent



Overview of ExecutionAgent



Overview of ExecutionAgent



Meta-Prompting

- Need technology-specific prompts:
  Guidelines for using specific languages and tools
- Option 1: Carefully craft prompts for tools that are widely used today
- Option 2: Ask LLM to generate guidelines

Meta-Prompting

- Need technology-specific prompts:
  Guidelines for using specific languages and tools
- Option 1: Carefully craft prompts for tools that are widely used today
- Option 2: Ask LLM to generate guidelines

Meta-Prompting

Example:
I want to install and run tests of a Java project. Give a list of 10 guidelines that I should follow to achieve my task (e.g., to figure out installation steps, dependencies, how to run tests) while considering latest practices and technologies.

Meta-Prompting

Response:

1. Review Project Documentation and README

Why: Documentation typically includes installation steps, dependencies, and setup instructions.

What to do: Check the README.md or other documentation files for prerequisites and specific instructions for installation and testing.

2. Verify Java Version Compatibility (...)

3. Check for Dependency Management Tool (...)

(...)

Agentic Feedback Loop

- **Step 1: Prompt asking for next command**
  - Contains history of past commands
- **Step 2: Invoke tool**
  - Available tools: File I/O, Linux terminal, `task_done`
- **Step 3: Summarize output and extract information**
  - Done by another LLM instance

If not successful after 40 cycles:

Restart from scratch (up to three total attempts)

Practical Challenges

- **Long-running and interactive commands**
  - After a timeout, show partial output to LLM and ask how to proceed
- **Special characters (color, progress bars, etc.)**
  - Heuristically clean terminal output
- **Checking if task done**
  - Ask for file with number of passed, failed, and skipped tests

Example Output

Dockerfile

```
FROM ubuntu:20.04
# Set the timezone to avoid interactive prompts
ENV TZ=XXX/YYY # omitted for anonymity
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ
# Update package lists and install necessary build tools and de
RUN apt-get update && \
apt-get install -y --no-install-recommends \
sudo \
build-essential \
liblzo2-dev \
libpam0g-dev \
liblz4-dev \
libcap-ng-dev \
libnl-genl-3-dev \
linux-libc-dev \
autoconf \
automake \
libtool \
cmocka-dev \
python3-docutils \
git \
&& apt-get clean && rm -rf /var/lib/apt/lists/* || exit 0
WORKDIR /app
# Clone the OpenVPN repository
RUN git clone https://github.com/OpenVPN/openvpn.git || exit 0
WORKDIR /app/openvpn
CMD ["bash"]
```

install_and_run.sh

```
#!/bin/bash
# Run autoconf to prepare the build system
autoreconf -fvi
# Configure the build system
./configure --enable-werror
# Compile the project
make -j$(nproc)
# Run tests
make check VERBOSE=1
echo "Setup and testing complete."
```

Test results

```
Total: 91
Pass: 85
Fail: 4
Skip: 2
```

Evaluation

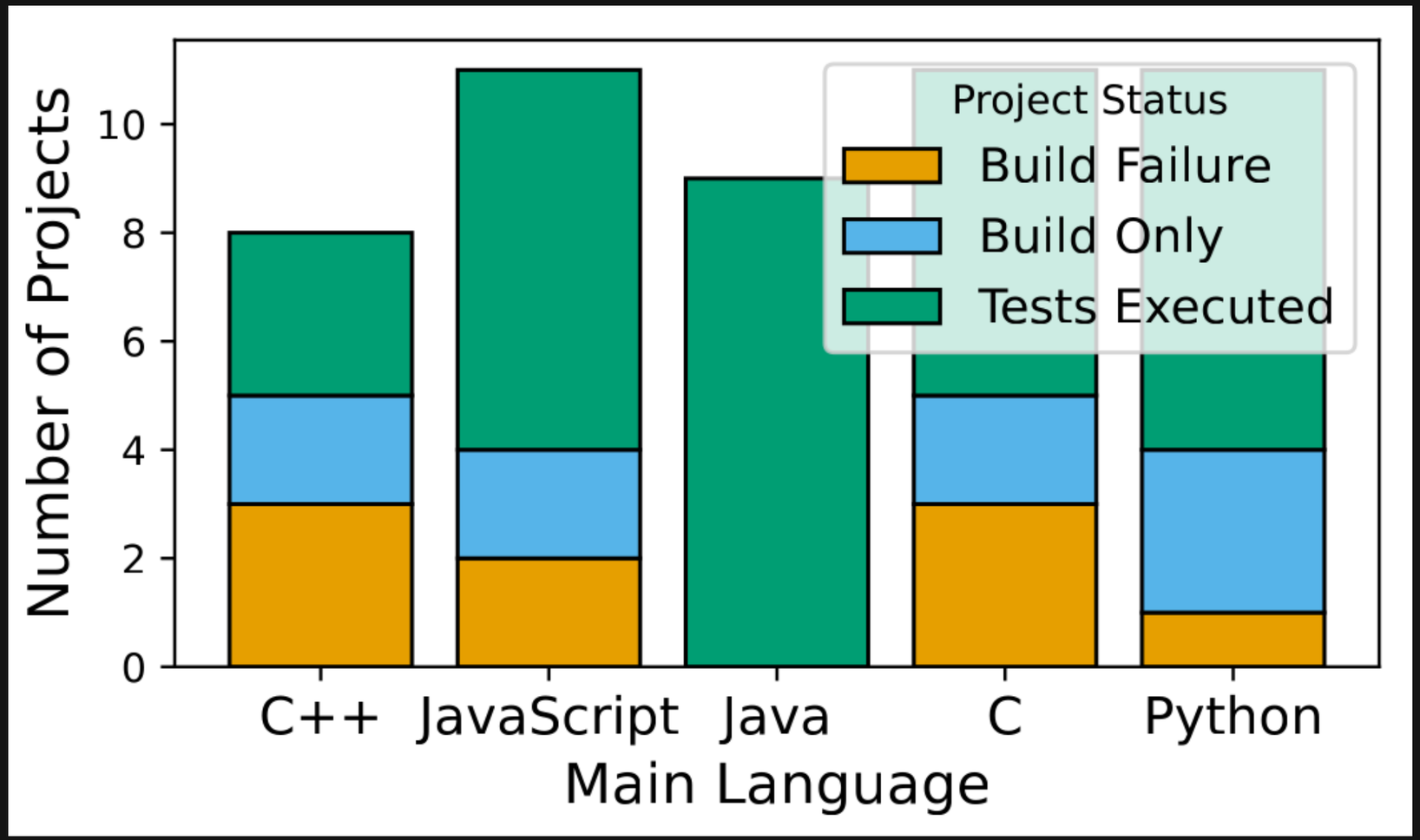
- **50 projects using 14 programming languages**
- **Ground truth of test execution results**
- **LLM: GPT-4o-mini**
- **Baselines**
  - LLM scripts: General-purpose scripts (one per language)
  - AutoGPT: General-purpose LLM agent
  - Flapy: Human-written script for Python projects

Effectiveness

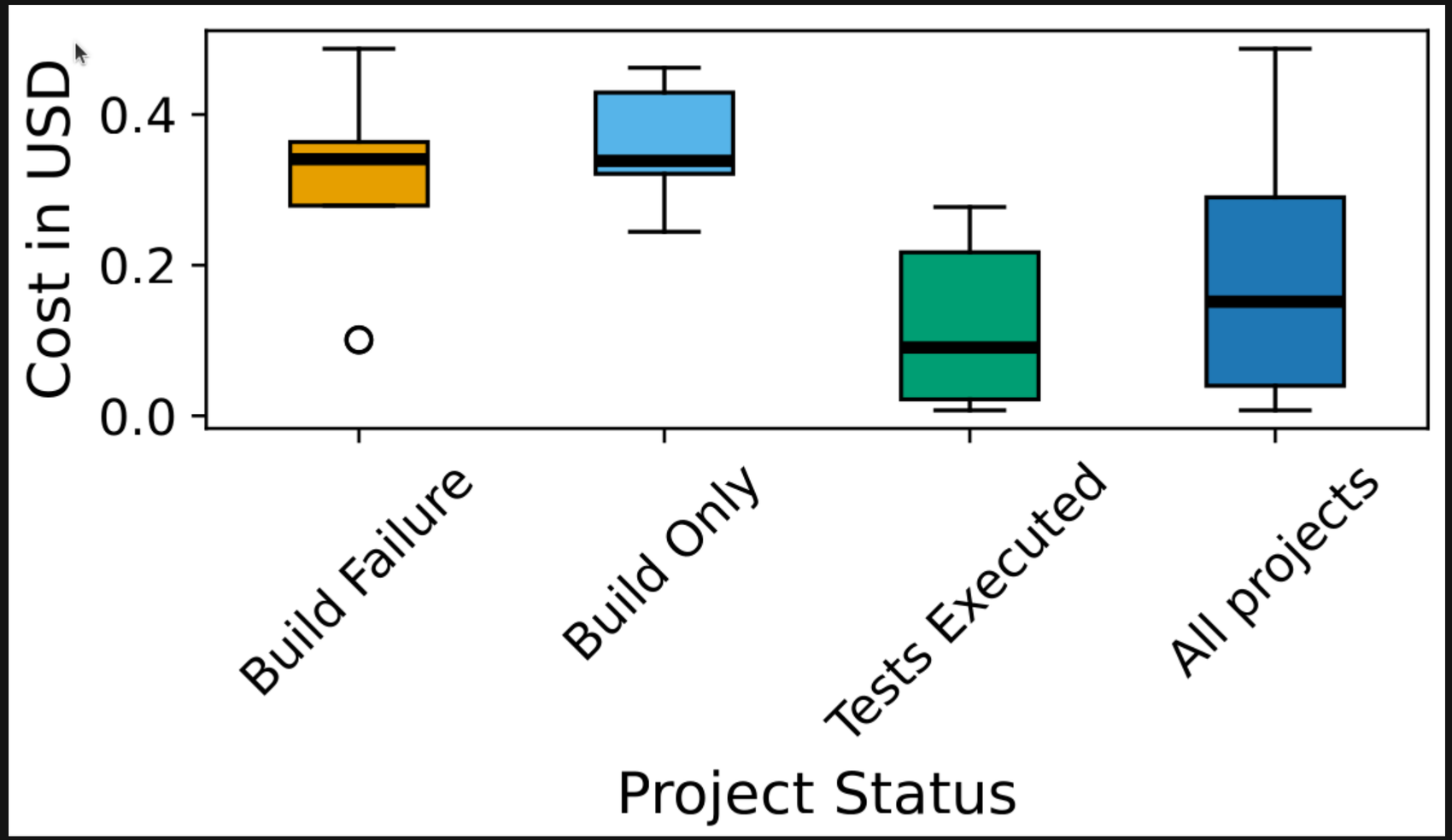
	Execution-Agent	LLM scripts	AutoGPT	Flapy
Built/installed	41 / 50	29 / 50	9 / 50	6 / 10
Executed tests	33 / 50	5 / 50	4 / 50	0 / 10
Results close to ground truth *	29 / 50	4 / 50	2 / 50	0 / 10

* Up to 10% deviation from ground truth numbers of passing/failing/skipped tests

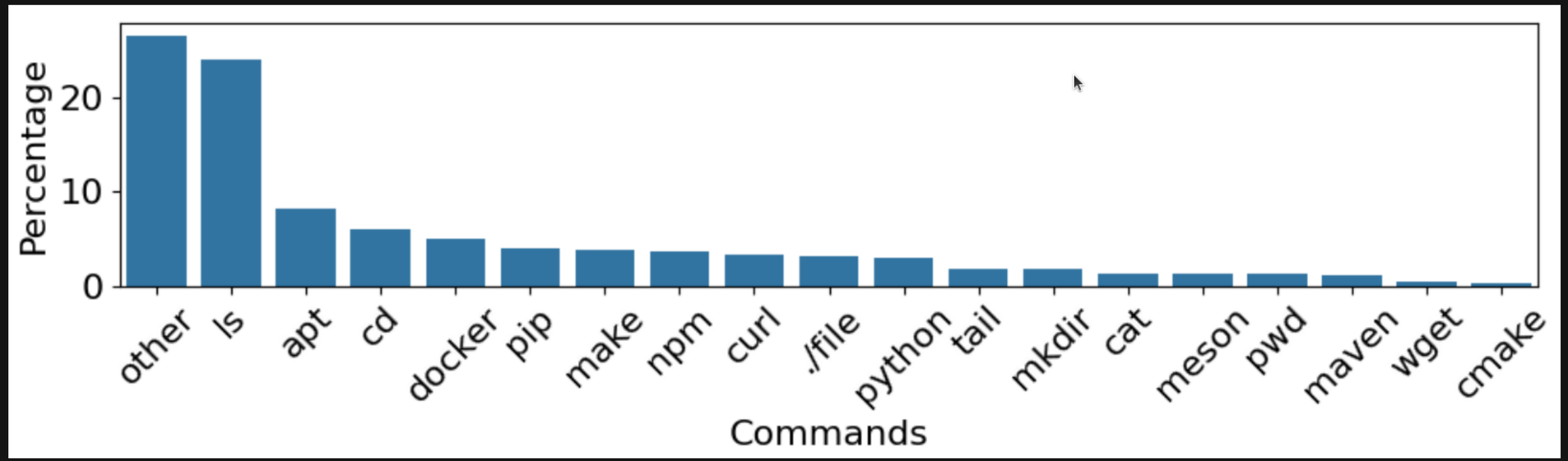
Effectiveness



Costs


Cost in USD
0.4
0.2
0.0
Build Failure
Build Only
Tests Executed
All projects
Project Status

Most Used Terminal Commands



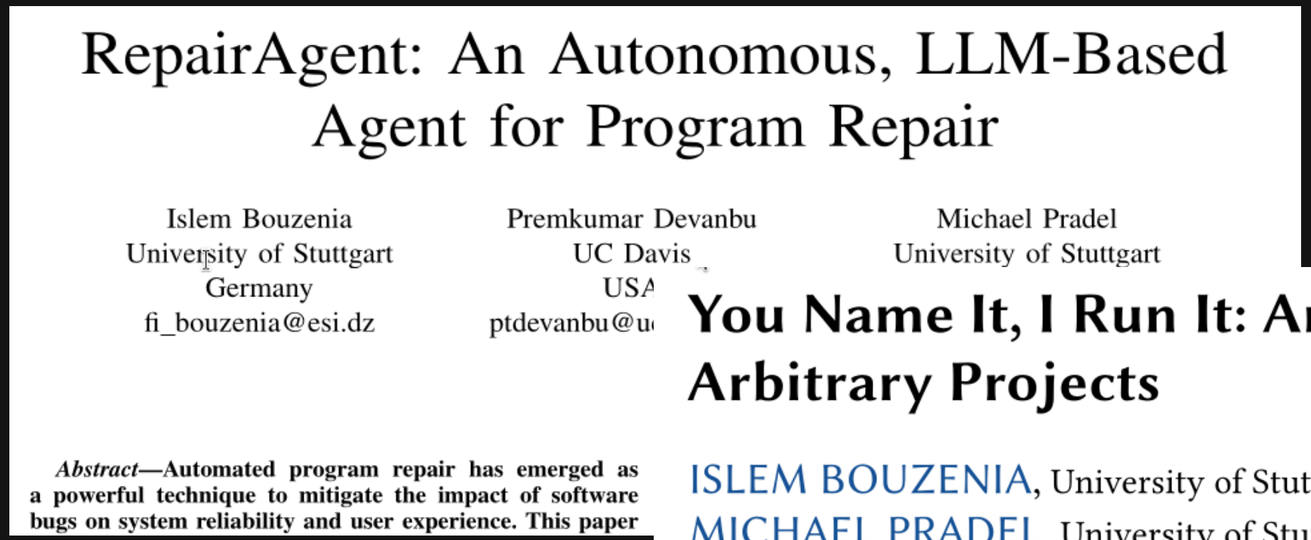
Average number of commands per produced script: 16

Open Challenges

- **Agent repeats the same mistakes**
  - E.g., try to use `sudo` even though it's not available
- **Agent fails to invoke "follow-up commands"**
  - E.g., after installing new version of GCC, should set it as default
- **Target platforms beyond Linux**

Conclusions

LLM agents: One step closer to human developers



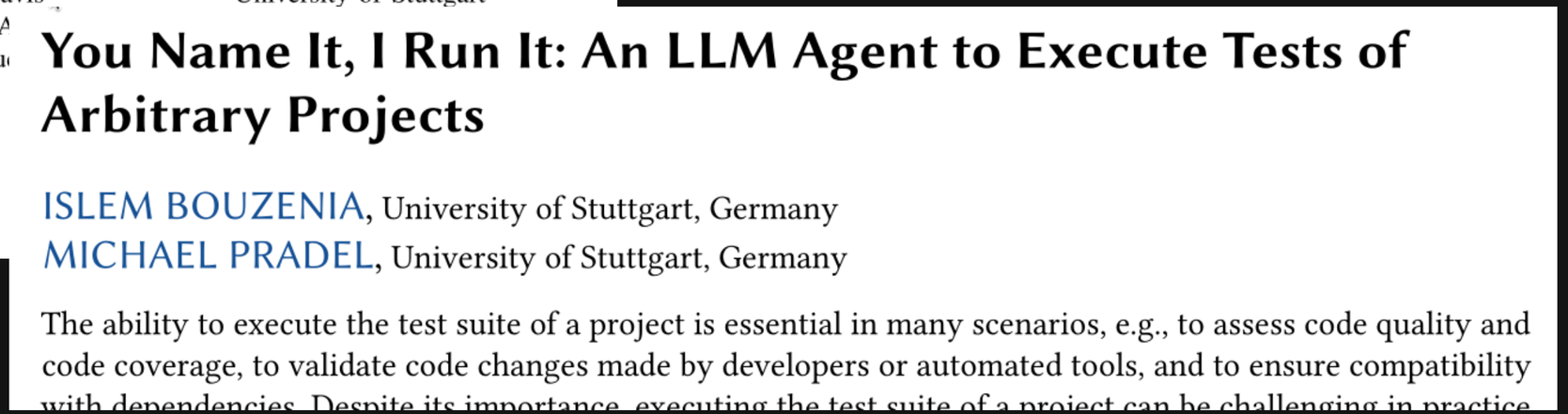
RepairAgent: An Autonomous, LLM-Based Agent for Program Repair

Islem Bouzenia
University of Stuttgart
Germany
fi_bouzenia@esi.dz

Premkumar Devanbu
UC Davis
USA
ptdevanbu@u

Michael Pradel
University of Stuttgart

Abstract—Automated program repair has emerged as a powerful technique to mitigate the impact of software bugs on system reliability and user experience. This paper



You Name It, I Run It: An LLM Agent to Execute Tests of Arbitrary Projects

ISLEM BOUZENIA, University of Stuttgart, Germany
MICHAEL PRADEL, University of Stuttgart, Germany

The ability to execute the test suite of a project is essential in many scenarios, e.g., to assess code quality and code coverage, to validate code changes made by developers or automated tools, and to ensure compatibility with dependencies. Despite its importance, executing the test suite of a project can be challenging in practice

https://github.com/sola-st/RepairAgent
https://github.com/sola-st/ExecutionAgent