Debugger-CLI:

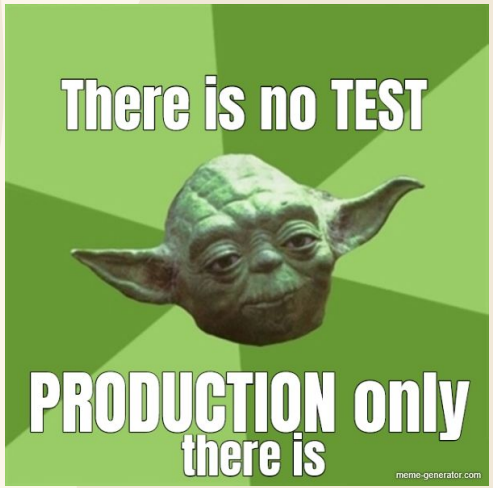
Automatic Test Code Generation CLI

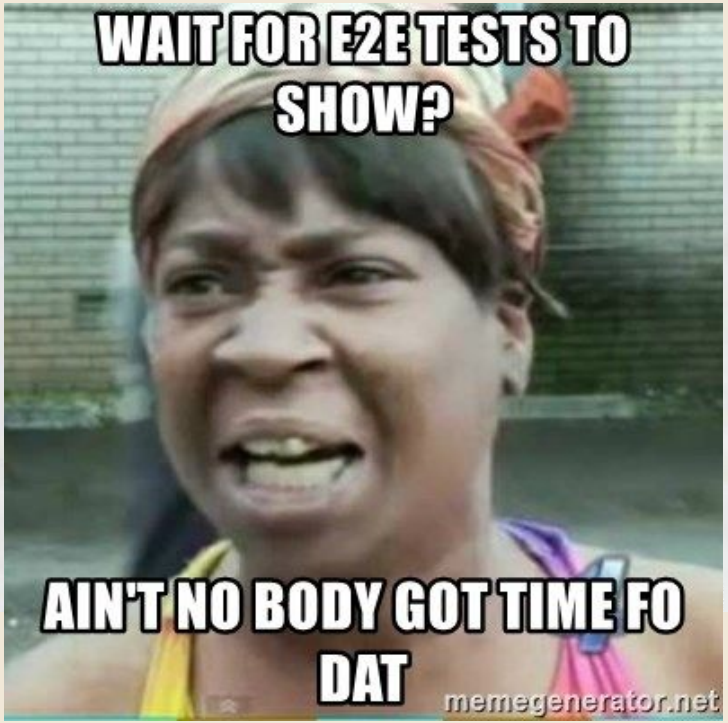
Team Debuggers (Team 4)

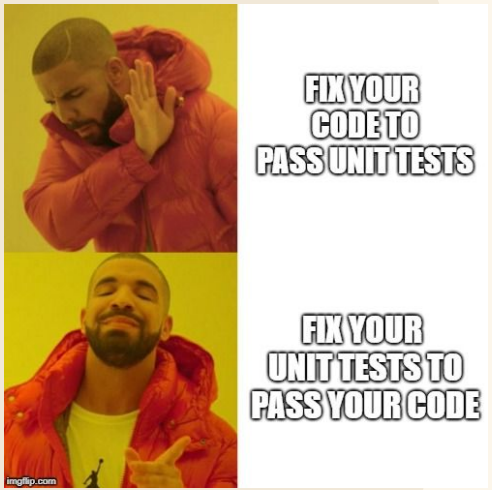
Jihun, Kristina, Akhdan, Minseong

1. Problem We want to solve

Developer lacks **time** for writing down








Finally, I've done with descent test codes!
DEV
V0.3.6
Email Sth
Pw Sth
Now, Please work on newly changed design, BTW we have only 2weeks..
cheer up!
PM
UX
design V1.00
DEV
uh oh.....
1 Day left....
Almost Done, But no time for test codes....
DEV
V0.9.9
em Sth Pw
Sth
D-Day
Dun! I need time for test codes!!
DEV
Nicely done!
But no time for test codes..
Let's work on new Updates!
PM
UX
V1.0
My test codes.....
DEV

It’s hard to write descent Test Code

Time and resources: E2E tests can be resource-intensive, often requiring significant time to write, run, and maintain. As a result, they can slow down the development cycle if not managed effectively.

Reducing human error: E2E testing is complex and requires meticulous attention to detail. Automating these tests reduces the risk of human error that can occur when testers become fatigued.

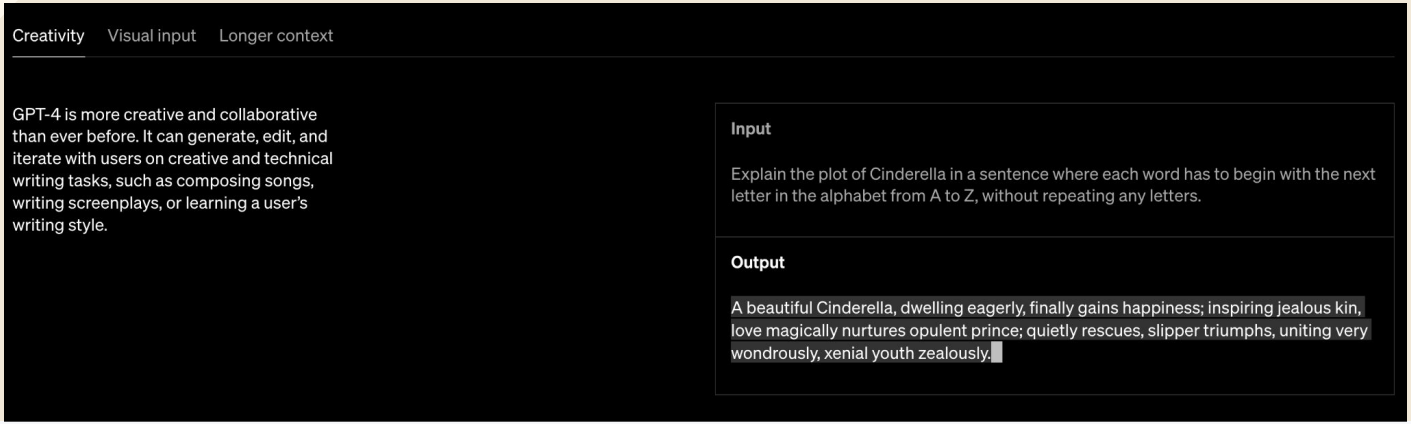
Maintainability: As the application evolves, E2E tests must be updated to reflect changes in the system. This ongoing maintenance can be challenging.

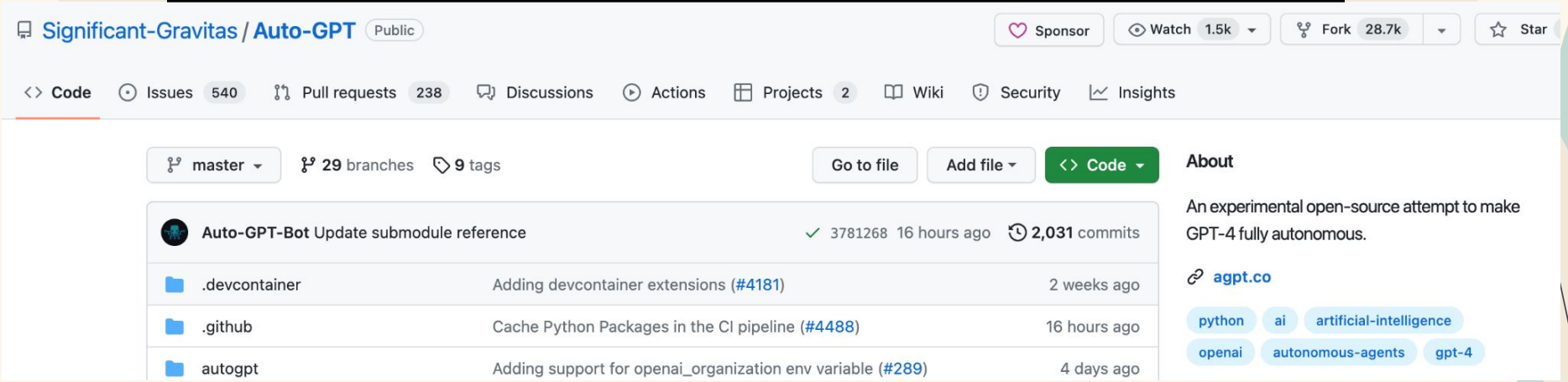
It’s hard to write descent Test Code

Test flakiness: E2E tests often interact with many layers of an application and its environment, leading to greater chances of test flakiness, or non-deterministic behavior, making the test results unreliable and the debugging process difficult.

Complex scenarios: E2E tests cover the entire software application from start to finish. This means designing tests that capture complex, real-world scenarios, which requires a deep understanding of the application and the business logic.

LLM Boom!


Creativity
Visual input
Longer context
GPT-4 is more creative and collaborative than ever before. It can generate, edit, and iterate with users on creative and technical writing tasks, such as composing songs, writing screenplays, or learning a user's writing style.
Input
Explain the plot of Cinderella in a sentence where each word has to begin with the next letter in the alphabet from A to Z, without repeating any letters.
Output
A beautiful Cinderella, dwelling eagerly, finally gains happiness; inspiring jealous kin, love magically nurtures opulent prince; quietly rescues, slipper triumphs, uniting very wondrously, xenial youth zealously.


Significant-Gravitas / Auto-GPT
Public
Sponsor
Watch 1.5k
Fork 28.7k
Star
Code
Issues 540
Pull requests 238
Discussions
Actions
Projects 2
Wiki
Security
Insights
master
29 branches
9 tags
Go to file
Add file
Code
Auto-GPT-Bot Update submodule reference
3781268 16 hours ago
2,031 commits
.devcontainer
Adding devcontainer extensions (#4181)
2 weeks ago
.github
Cache Python Packages in the CI pipeline (#4488)
16 hours ago
autogpt
Adding support for openai_organization env variable (#289)
4 days ago
About
An experimental open-source attempt to make GPT-4 fully autonomous.
agpt.co
python
ai
artificial-intelligence
openai
autonomous-agents
gpt-4

LLM Boom!

By utilizing this GPT’s brilliant power, we can automate complex and repetitive and faulty process with test code generation agent.

Using LLM iteratively to reduce human workload is feasible and useful.

What we want to build as a final outcome is GPT automatically build test codes and run iteratively as developers creates more codes.

Technique Ideation Process - Jihun

Main Topic: Automated Test Case Generation

Let developers to add some metadata that can be used in automated test case generation -> for login page, explicitly add some x-path tag, etc..

Professor feedback: Why even developer should do some extra work to utilize your tool? Why don't it become fully automated?

Paper suggested by professor (Context-aware Automated Text Input Generation for Mobile GUI Testing)

-> For Android app there's some feature that mobile gui can be converted into xml format, and the researchers feed that data into GPT model so that the model can create context-aware text inputs which could be used for GUI Testing scenarios.

We redirected to utilize GPT model's strong context capturing feature.

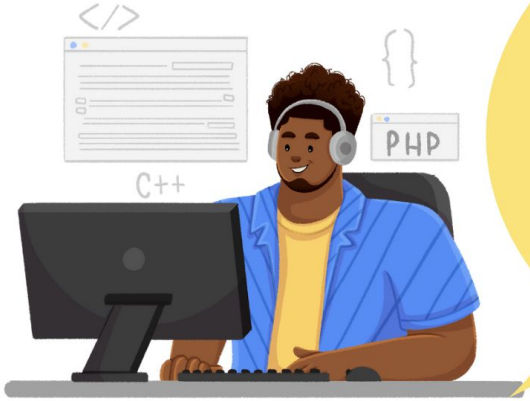
Why don't we just fully automate utilizing GPT(GPT scan the page and create some testing codes)

But, this could be overwhelming and quite dangerous. So, let developer can explicitly define their testing needs and prompt it to GPT with code base information could achieve reasonable test case generations.

2. Technique Ideation Process

Previous logic : Developer put yaml information about test cases to create test codes.
Improved logic : Our software analyze every codebase and create multiple e2e test scenarios automatically. Then it creates 5~10 test cases based on generated scenarios.

2.1. Previous Idea



HTML Format looks like : blabla

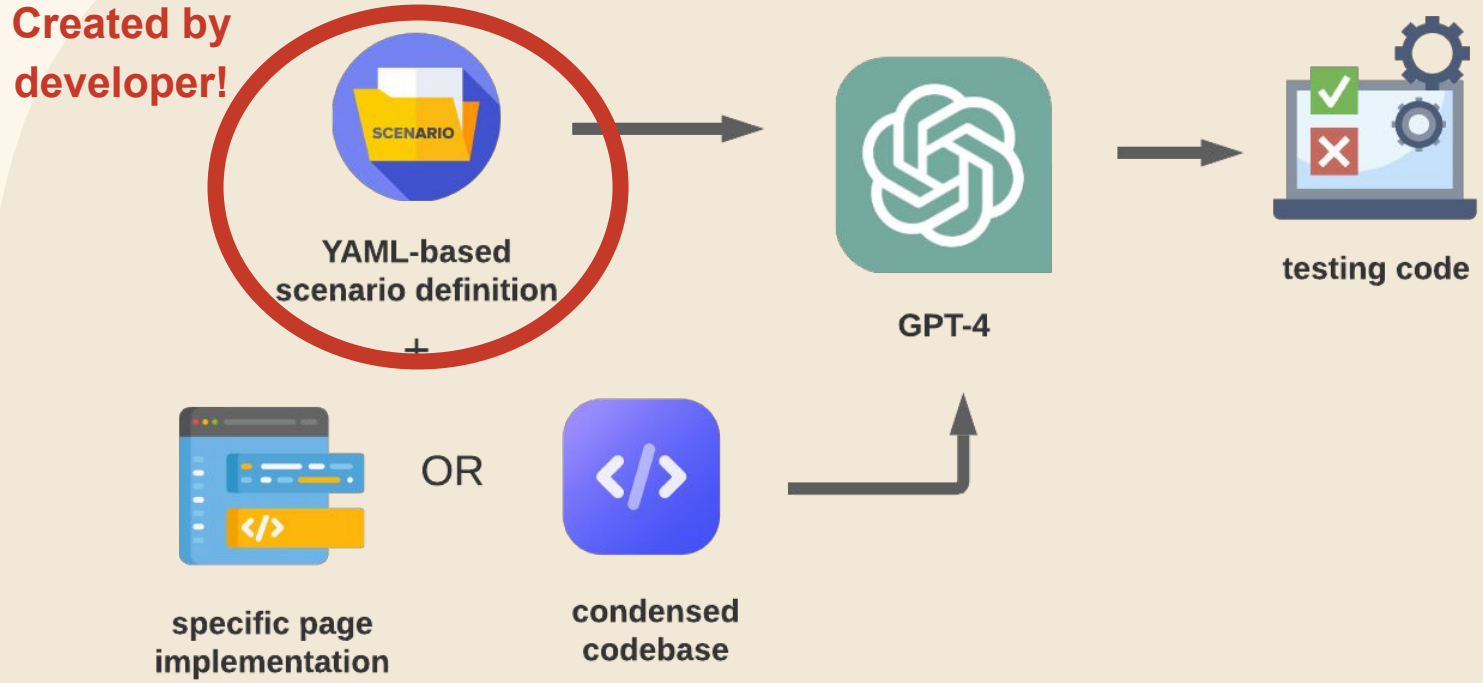
login-page test :
#email
- NOT_EMPTY

#password
- NOT_EMPTY NUMBER
- OF SPECIAL_CHARACTERS >= 2

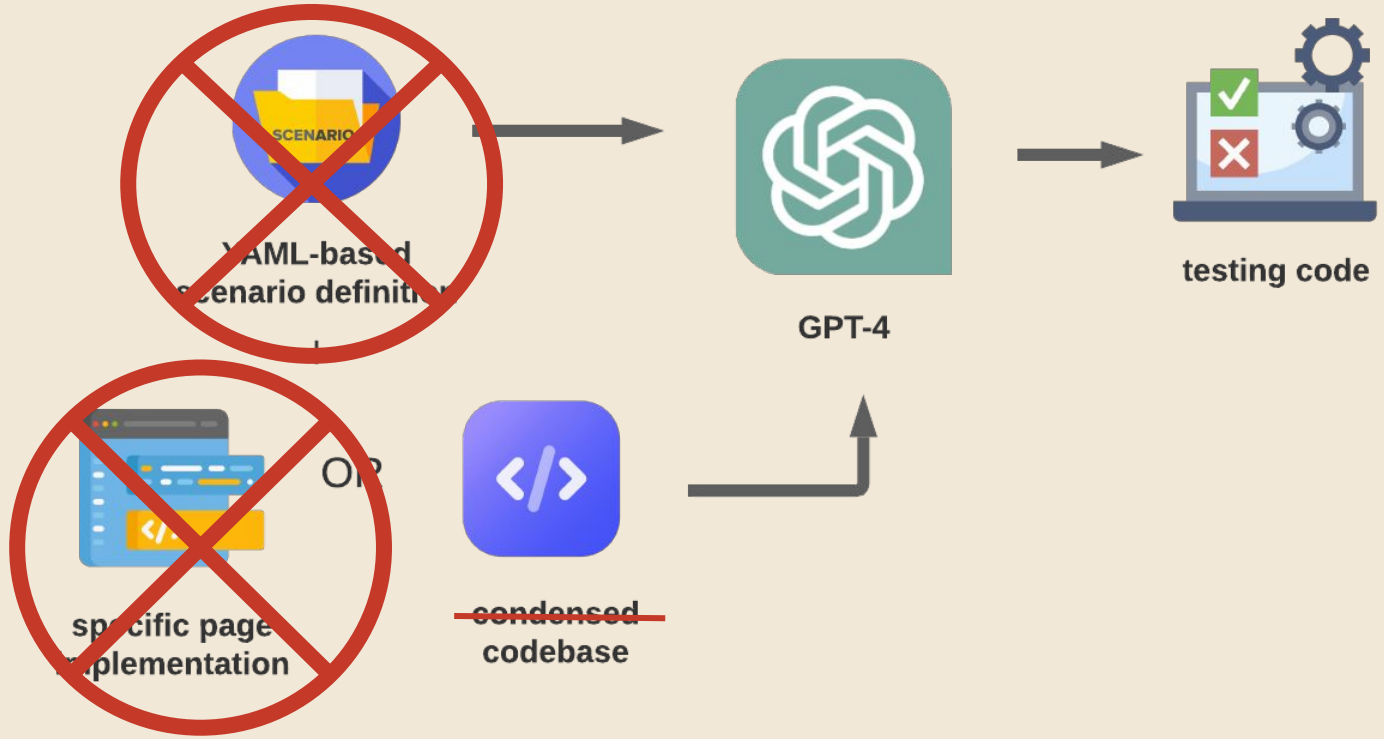
action : button.signup click
result : redirected

Developer puts metadata ➡ Creates E2E test cases automatically

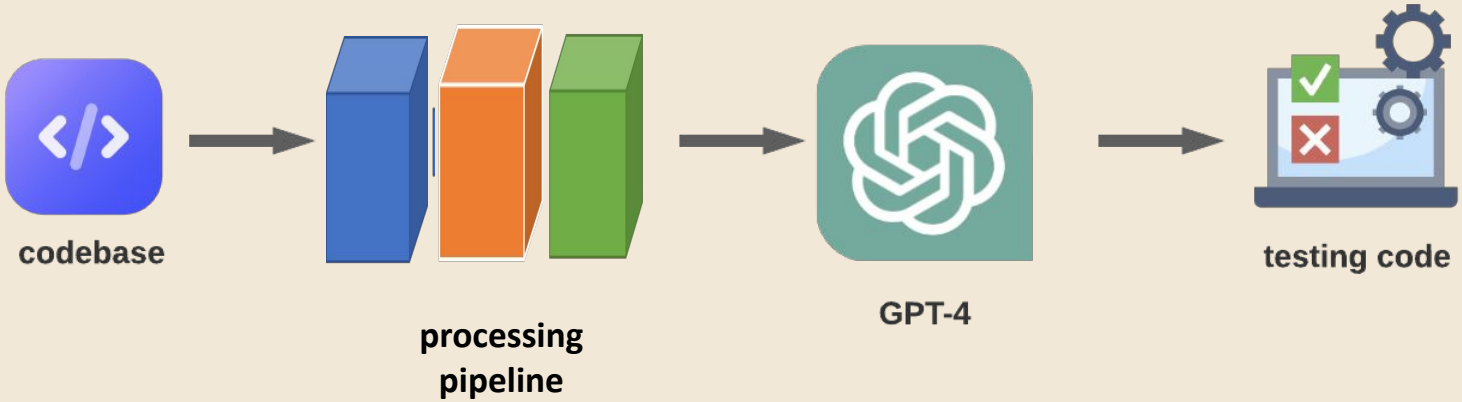
2.1. Previous Idea



2.2 Improved Idea

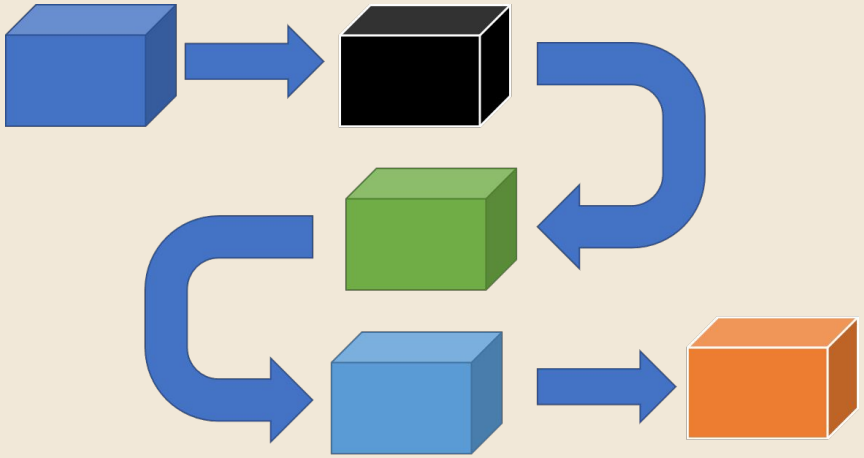


2.2 Improved Idea



1. Our software processes the entire codebase in multiple stages
2. E2E test scenarios are generated
3. Test codes are generated based on provided scenarios

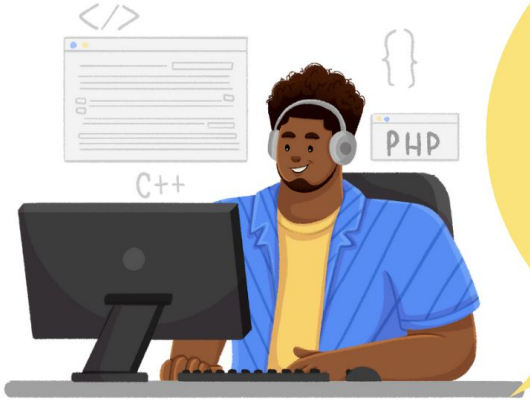
2.1. Main Topic



End to End Testing

Automated E2E Test Case Generation

2.2. Initial Idea



HTML Format looks like : blabla

login-page test :
#email
- NOT_EMPTY

#password
- NOT_EMPTY NUMBER
- OF SPECIAL_CHARACTERS >= 2

action : button.signup click
result : redirected

Developer puts metadata ➡ Creates E2E test cases automatically

2.3. Feedback from Professor

Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing

1st Zhe Liu
Institute of Software
Chinese Academy of Sciences,
Beijing, China
liuzhe181@mails.ucas.edu.cn

2nd Chunyang Chen
Monash University,
Melbourne, Australia
Chunyang.chen@monash.edu

3rd Junjie Wang
Institute of Software
Chinese Academy of Sciences
**Corresponding author*
junjie@iscas.ac.cn

4th Xing Che,
5th Yuekai Huang
Institute of Software
Chinese Academy of Sciences,
Beijing, China

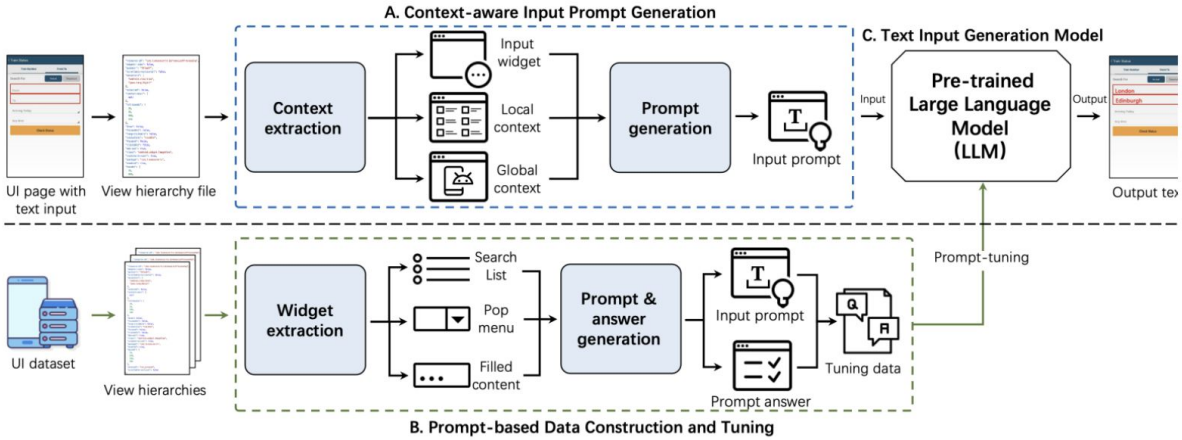
6th Jun Hu
Institute of Software
Chinese Academy of Sciences,
Beijing, China
hujun@iscas.ac.cn

7th Qing Wang
Institute of Software
Chinese Academy of Sciences
**Corresponding author*
wq@iscas.ac.cn

***Abstract*—Automated GUI testing is widely used to help ensure the quality of mobile apps. However, many GUIs require appropriate text inputs to proceed to the next page, which remains a prominent obstacle for testing coverage. Considering the diversity and semantic requirement of valid inputs (e.g., flight departure, movie name), it is challenging to automate the text input generation. Inspired by the fact that the pre-trained Large Language Model (LLM) has made outstanding progress in text generation, we propose an approach named `QTypist` based on LLM for intelligently generating semantic input text according to the GUI context. To boost the performance of LLM in the mobile**

explore mobile apps by executing different actions such as scrolling, clicking based on the analysis of code structure of the current page to verify UI functionality. However, most of them focus on exploration algorithm improvement, while few of them are concerned with the complicated interaction with app like *text input* generation. According to our observation in Section II, most apps have some pages requiring specific text inputs to go to the next page. As seen in Fig 1, without correct information for flight searching, the consecutive pages

2.3. Feedback from Professor



1. Convert Mobile GUI into XML Format
2. Train a GPT model with XML Data
3. Build Automated-Testing with GPT

2.4. Pivot the idea

Why don’t we just fully automate utilizing GPT?

1. GPT scan the page (html, jsx, routes, etc)
2. Automatically create test codes
3. Done!

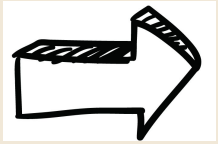
2.4. Fully-Automated Testing Framework

Huge Risk & Quite Dangerous Approach

Q: Why?
A: Developers won't get result as they expected

2.4. Fully-Automated Testing Framework

1. Prompt their testing needs with specific format
2. Write their needs in detail



Get Reasonable Test Codes

The technique the team is proposing - Kristina

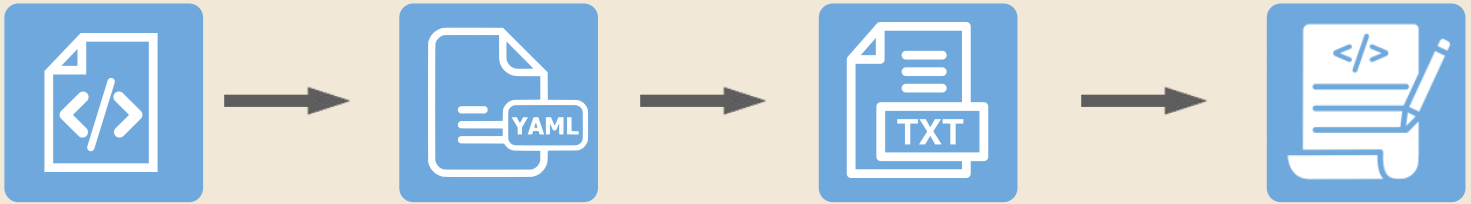
- yaml based test scenario definition

- feed it into GPT-4 along with either the condensed version of the codebase (e.g., using the OpenAI emdedding_utils) or the specific implementation to automatically create testing codes which meet the needs from the definition (Use Jest, Faker for testing code)

- If we use the entire code base for GPT analysis, we could also add the functionality to also suggest possible test scenarios

3. Proposed technique

Design Overview



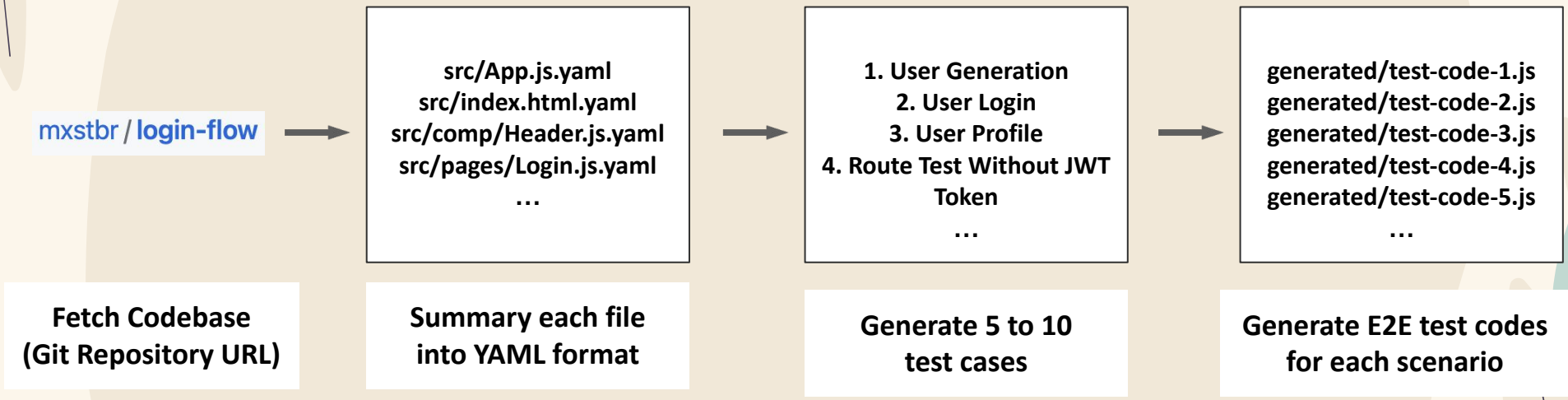
Fetch Codebase
(Git Repository URL)

Summary each file
into YAML format

Generate 5 to 10
test cases

Generate E2E test codes
for each scenario

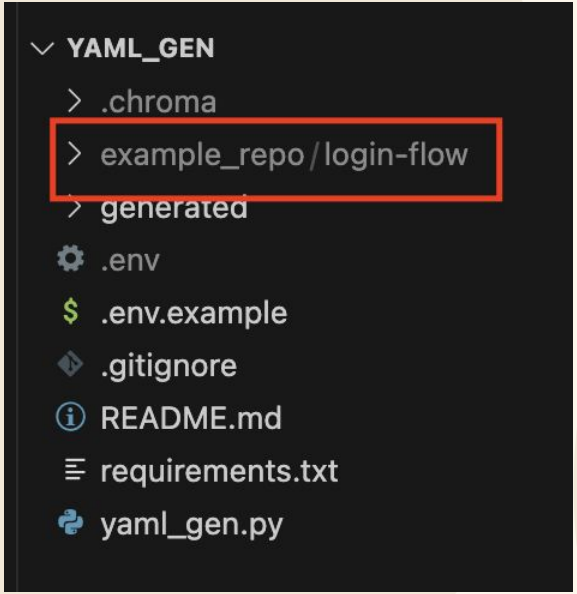
Design Overview



Fetch Codebase

```
from langchain.document_loaders import GitLoader
loader = GitLoader(repo_path="./example_data/login-flow/",
branch=branch).load()
```

Python Langchain GitLoader



Summary Each File

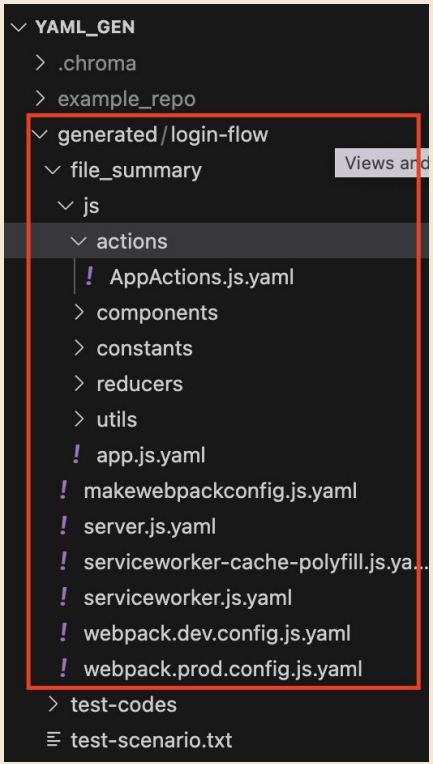
code
(js,jsx,ts,html,etc)

↓

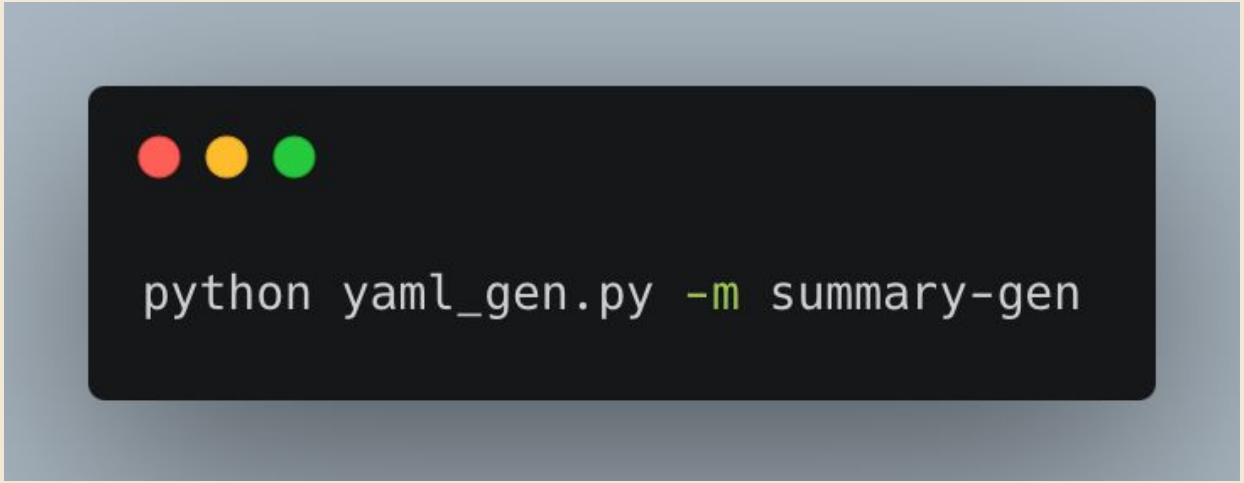
summarize with gpt-3

↓

YAML summary output



Generate Summary YAML CLI



```
name: AppActions.js
location: js/actions/
description: This file contains the actions ...
dependencies:
  - redux
  - auth from ../utils/
  - ...
functions:
  - login(username, password)
    - description: Logs in a user with the given username and password.
    - parameters:
      - username: The username of the user to be logged in. (string)
      - password: The password of the user to be logged in. (string)
    - returns: A dispatch function that sends out actions.
  - logout()
    - description: Logs out the current user.
    - returns: A dispatch function that sends out actions.
  - register(username, password)
    ...
```

Generate Test Scenarios

Vector Store
(based on YAML summary files)

↓

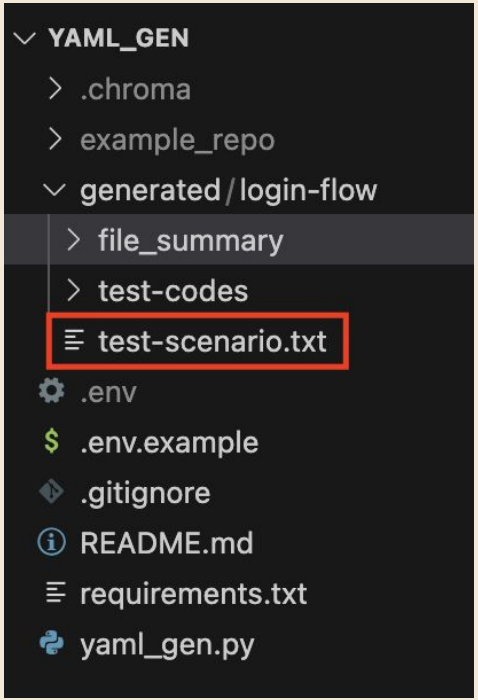
OpenAI Embedding

↓

Generate 30 Test Cases

↓

Choose 5 to 10 out of them



Generate Test Scenario CLI

1. User registration:
   - Test a new user registration process with valid credentials and ensure successful account creation.
   - Test user registration with an existing email address or username, and ensure the system prompts an error message.
   - Test user registration with invalid email formats and ensure the error message is displayed.

2. User login:
   - Test the user login process with valid credentials and ensure successful login.
   - Test the user login with incorrect credentials, and ensure the system prompts an error message.

3. User profile:
   - Test updating user profile information and ensure the changes are saved successfully.
   - Test updating a user's profile with invalid data formats, and ensure the system prompts an error message.

4. ...

Generate E2E Test Codes

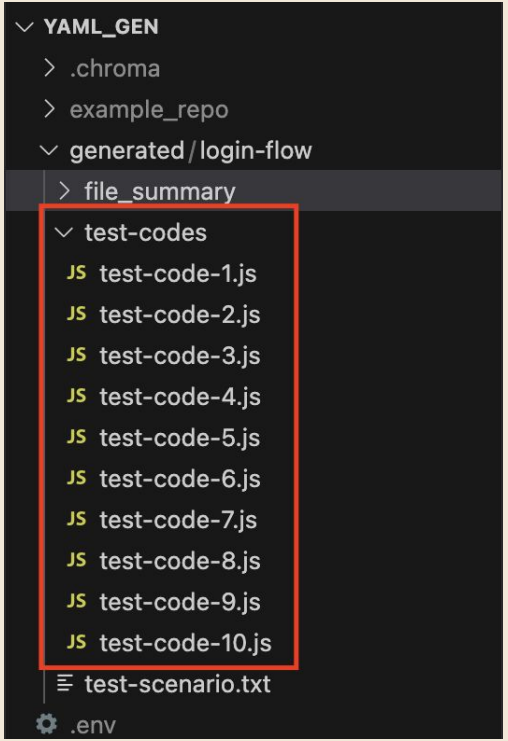
Test scenarios

↓

GPT-4 Generate test codes

↓

5 to 10 E2E Test Codes Generated



Generate Test Codes CLI

```
const { register } = require("../js/utils/auth");

describe("User Registration", () => {
  test("successful user registration with valid credentials", async () =>
{   const newUser = {
      username: "testuser",
      email: "testuser@example.com",
      password: "password123",
    };
    const response = await register(
      newUser.username,
      newUser.email,
      newUser.password
    );

    expect(response.success).toBe(true);
    expect(response.message).toBe("Registration successful");
  });

  test("user registration with existing email or username", async () => {
    const existingUser = {
      username: "testuser",
      email: "testuser@example.com",
      password: "password123",
    };
...
```

https://github.com/xMHW/debuggers-cli

README.md

Automatic Test Code Generation CLI

This CLI tool is designed to assist in generating test scenarios and test codes for a React application. It uses OpenAI's language model to generate test-related content based on provided inputs and the repository's source code.

Prerequisites

Before using this tool, make sure you have the following:

- A public GitHub repository containing the React application.
- An OpenAI API key.
- Python 3.6 or above installed.

Setup

1. Clone this repository to your local machine.
2. Install the required Python packages by running the following command in the project directory:

```
pip install -r requirements.txt
```

3. Rename the `.env.example` file to `.env` and update the following values:

```
OPENAI_API_KEY=[YOUR OPENAI API KEY]
GIT_REPOSITORY=[GIT REPOSITORY URL]
TEST_SCENARIO_COUNT=[NUMBER OF TEST SCENARIOS TO GENERATE]
```

YAML-based scenario definition

```yaml
tests:
    - it:
        name: it-login-success
        actions:
          - navigate:
              path: /login
              file-path: src/pages/login.html
          - setInputValue:
              selector: "#email"
              value: Email Input With Less Than 30 Letters
          - setInputValue:
              selector: "#password"
              value: Password Input With Less Than 20 Letters Including At Least
"!@#$%^&*" Characters
          - sleep: 5s
          - clickButton:
              selector: "body > div > div > .button"
        expected:
          - jwt-token: "Authorization" Header Should Not Be Empty
          - redirected-page: /main
```

YAML-based
scenario definition

```yaml
- it:
    name: it-login-fails
    actions:
      - navigate:
          path: /login
          file-path: src/pages/login.html
      - setInputValue:
          selector: "#email"
          value: Email Input With More Than 30 Letters
      - setInputValue:
          selector: "#password"
          value: Password Input With Less Than 20 Letters Only With Alphabet
      - sleep: 5s
      - clickButton:
          selector: "body > div > div > .button"
    expected:
      - jwt-token: "Authorization" Header Should Be Empty
      - redirected-page: None (stay in /login page)
      - popup:
          selector: "#login-failed"
```

specific page
implementation

OpenAI
embeddings_utils

entire
codebase

condensed
codebase

test scenario
suggestions?

- Feed scenario definition & code into GPT-4 to automatically create testing codes which meet the needs from the definition

  “Please make javascript test code based on below yaml file”.

```
const faker = require("faker");
const { test, expect, describe } =
require("@jest/globals");
const {
  navigate,
  setInputValue,
  sleep,
  clickButton,
  getAuthorizationHeader,
  getCurrentPagePath,
  isPopupVisible,
} = require("./utils");
```

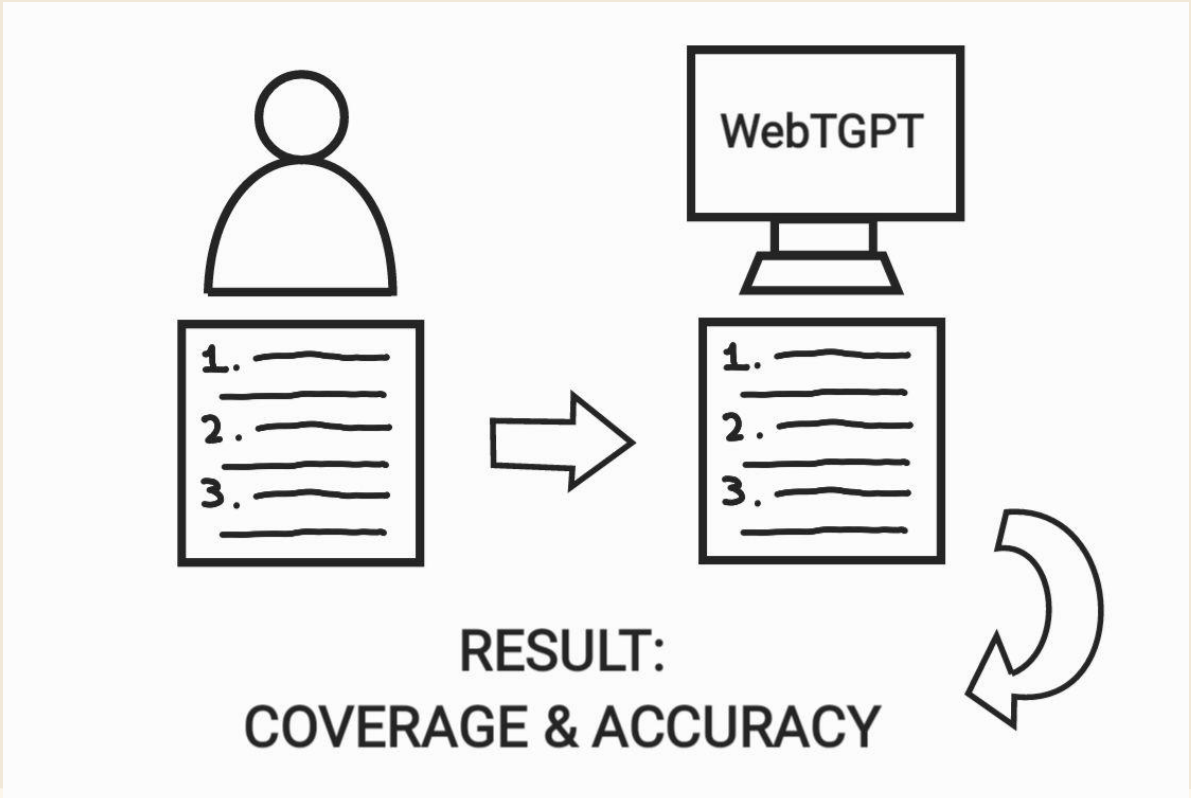
```
describe("test main page with login and check token & redirect", () => {
  it("it-login-success", async () => {
    await navigate("/login", "src/pages/login.html");

    const email = faker.internet.email().slice(0, 30);
    const password = faker.internet.password(20, true,
/[\!\@\#\$\%\^\&\*]/);
    await setInputValue("#email", email);
    await setInputValue("#password", password);

    await sleep(5000);

    await clickButton("body > div > div > .button");

    const jwtToken = getAuthorizationHeader("Authorization");
    expect(jwtToken).toBeTruthy();

    const redirectedPage = await getCurrentPagePath();
    expect(redirectedPage).toBe("/main");
  });
```

```
  it("it-login-fails", async () => {
    await navigate("/login", "src/pages/login.html");

    const email = faker.internet.email().repeat(2);
    const password = faker.internet.password(20, false);

    await setInputValue("#email", email);
    await setInputValue("#password", password);

    await sleep(5000);

    await clickButton("body > div > div > .button");

    const jwtToken = getAuthorizationHeader("Authorization");
    expect(jwtToken).toBeFalsy();

    const redirectedPage = await getCurrentPagePath();
    expect(redirectedPage).toBe("/login");

    const isLoginFailedPopupVisible = await isPopupVisible("#login-
failed-pop");
    expect(isLoginFailedPopupVisible).toBeTruthy();
  });
});
```

4. Evaluation Method

- Prev: 2 method
- Curr: Choose human testing method
- Assumption: We assumed that we have access to test scenarios and test codes (If not, we will have to manually create them)
- Coverage: Compare scenarios.txt with real scenarios. This one has to be manually checked by human.
- Accuracy: Compare the results of generated codes and hand-written codes
- Flexibility: Test more kinds of websites
- Limitations: Some generated codes assume ./js/utils/* like auth.js, fakeServer.js, and fakeRequest.js.

4.1. Chosen Method: Comparison with human written tests



4.2.0. Assumption

- For each website, we assumed that we have access to pre written test scenarios and test codes as the upper bound (If not, we will have to manually create them)

4.2.1. Testing Coverage

1. User registration:
   - Test a new user registration process with valid credentials and ensure successful account creation.
   - Test user registration with an existing email address or username, and ensure the system prompts an error
   - Test user registration with invalid email formats and ensure the error message is displayed.

2. User login:
   - Test the user login process with valid credentials and ensure successful login.
   - Test the user login with incorrect credentials, and ensure the system prompts an error message.

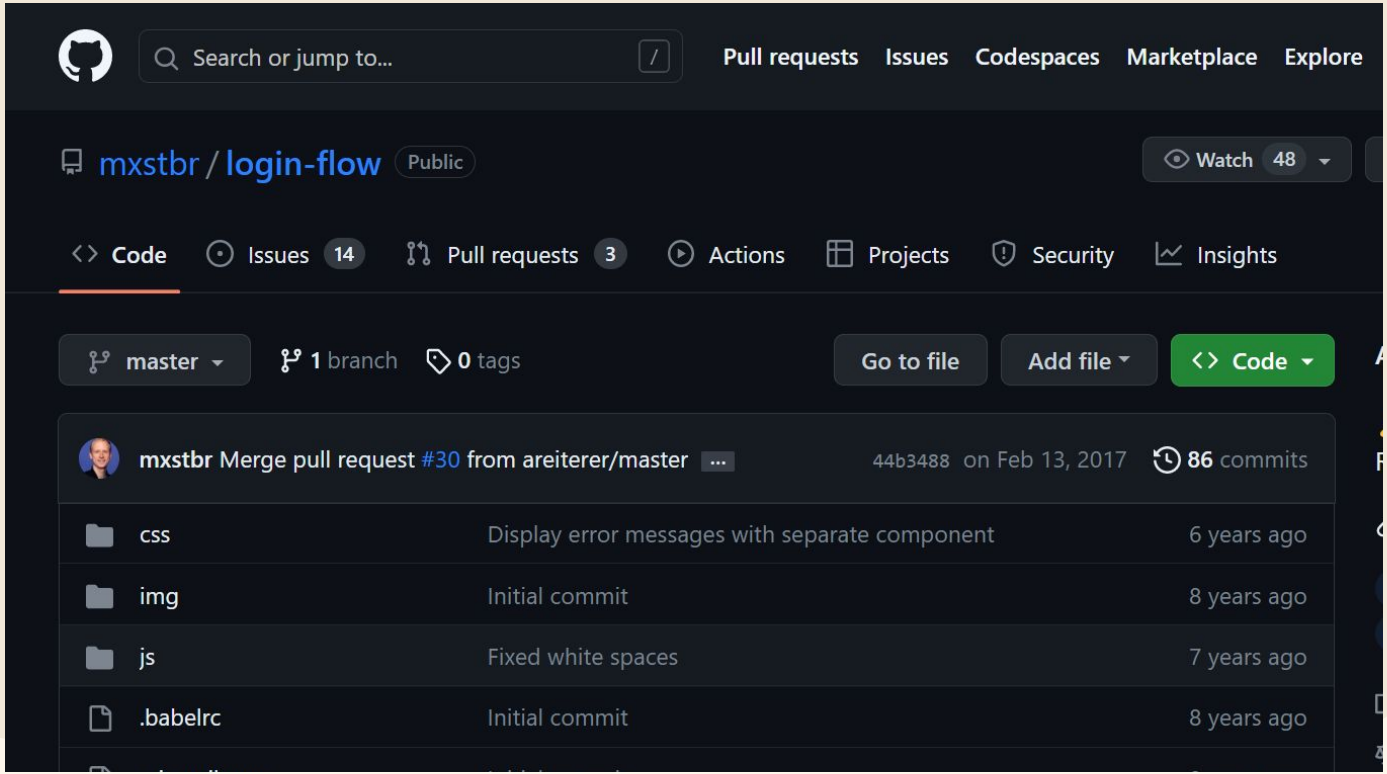
3. User profile:
   - Test updating user profile information and ensure the changes are saved successfully.
   - Test updating a user's profile with invalid data formats, and ensure the system prompts an error message.

4. Project creation:
   - Test the process of creating a new project with mandatory fields filled, and ensure the project is created
   - Test creating a project while leaving mandatory fields blank, and ensure an error message is displayed.

4.2.2. Testing Accuracy

```
/**
 * E2E Test Scenario 3: User Profile
 *    - Test updating user profile information and ensure the changes are saved
 *    - Test updating a user's profile with invalid data formats, and ensure th
 */

const fakeAuth = require("./js/utils/auth");
const fakeRequest = require("./js/utils/fakeRequest");
const { fakeServer } = require("./js/utils/fakeServer");

// Set up and start fake server
fakeServer.init();

// Test updating user profile information and ensure the changes are saved suc
test("Update user profile information with valid data", async () => {
  const username = "testuser";
  const password = "testpassword";

  // First, register a new user
  await new Promise((resolve) => {
    fakeAuth.register(username, password, resolve);
  });
```

4.2.3. Testing Flexibility



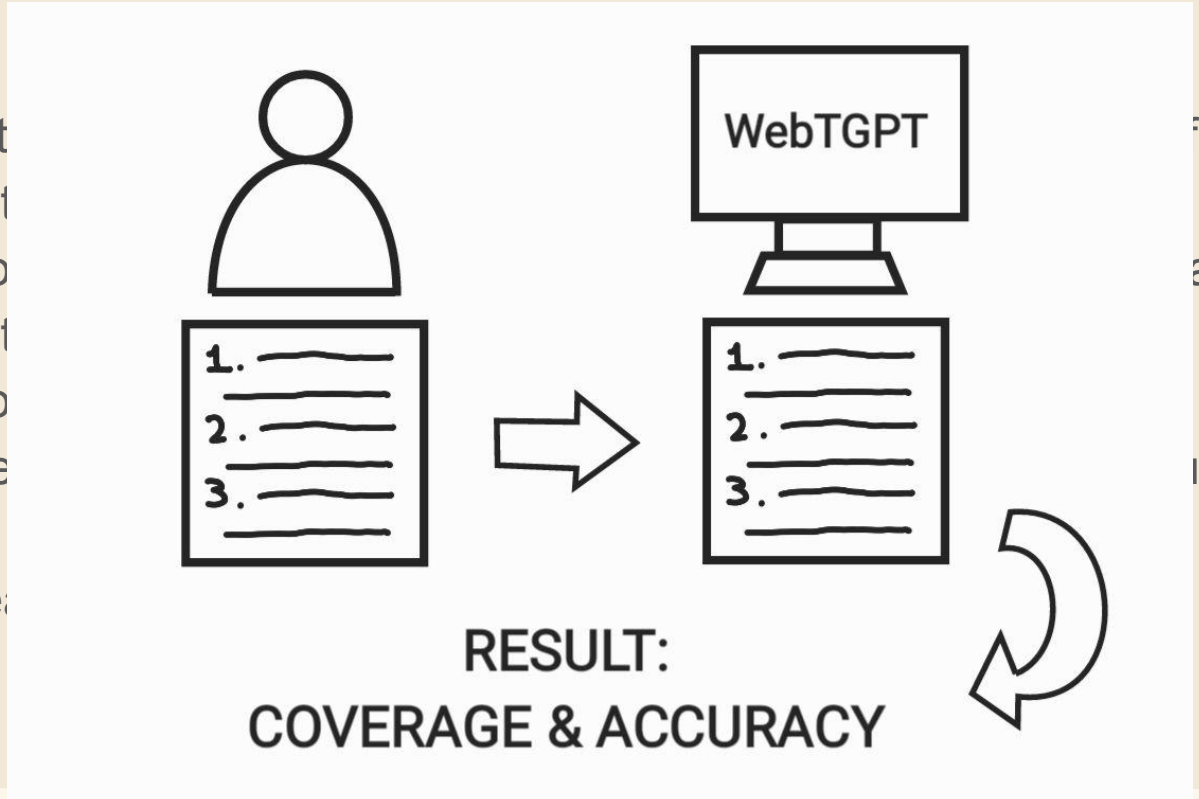
Ways to evaluate the correctness and competence of the technique

4.1. Comparison with human written tests

4.2. F-score measure for the conformity of test cases

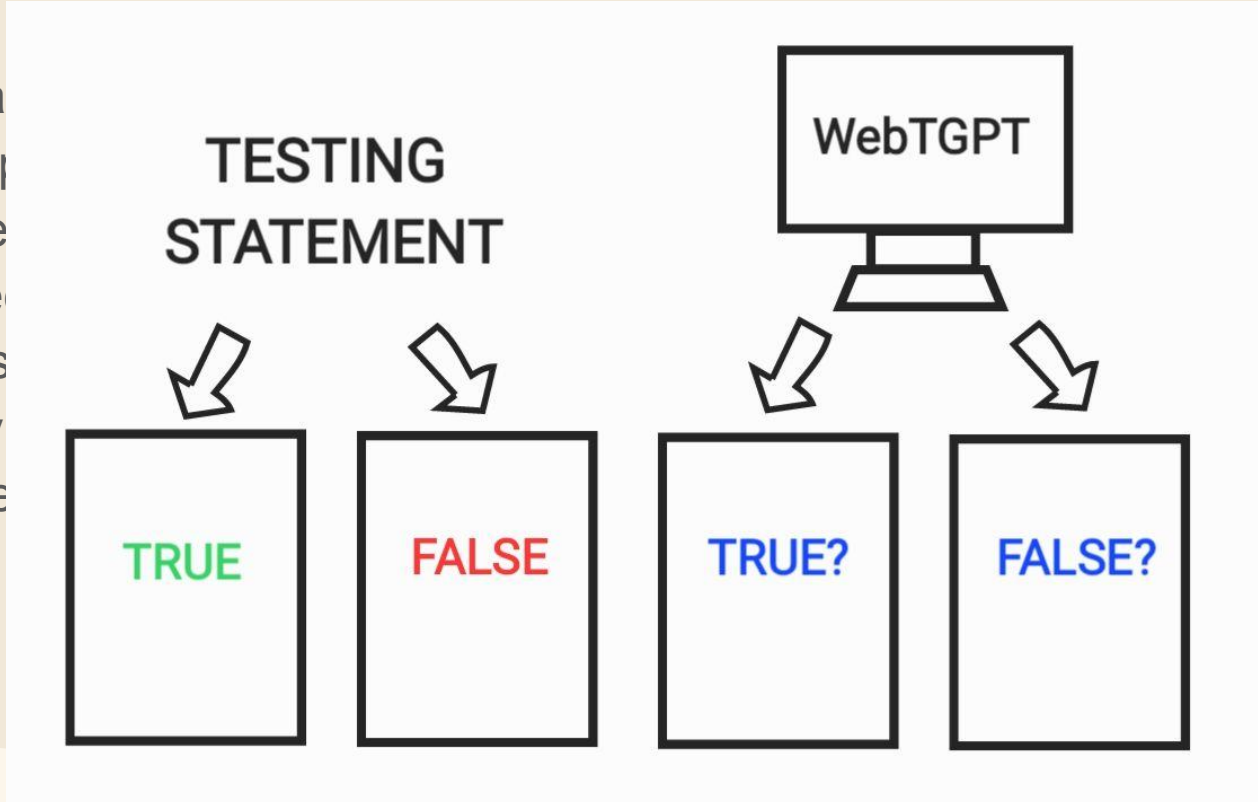
4.1. Comparison with human written tests

- Set
  edit
- Pro
  test
- Pro
- Cre
  for
- Me



4.2 F-score measure for the conformity of test cases

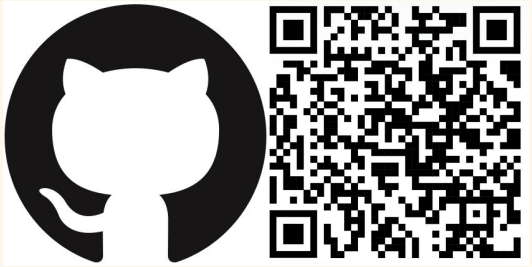
- Ba
  sep
  we
- Fe
  tes
- By
  me



Any Questions?

Check this out!

https://github.com/xMHW/debuggers-cli



Team4

(Debugger-CLI : Automatic Test Code Generation CLI)

A way of evaluation to show the proposed technique works and is competent - Dzaky

- Setup various testing scenarios that could be regarded as quite generally everywhere(signup, signin, profile edit.. etc)
- Write down human generated testing code.(This could be the upper boundary of testing performances.)
- And our “WebTGPT” generated testing code.
- Create 10 test suites(with possible faults for specific test statement from the yaml file) and measure the test accuracy.
- Compare whether the both test covers that case.

A way of evaluation to show the proposed technique works and is competent

- Based on testing statement we feed to “WebTGPT”(let’s say, password text input should not be longer than 18 chars.), we locate the websites which conform this statement and also other websites which don’t conform so that this could be separated into True and False case for the statement.
- Feed “WebTGPT” the implementation of the webpages in list(html or converted infos about that page) and the testing statement so that model can create the testing code.
- By running the testing codes generated by the “WebTGPT”, we measure the F-Score for the performance of our technique.

1. Problem definition

Causes of Depression

Interpersonal relationship

PTSD

Mental health

loneliness

Alcohol & drugs

Sleep, diet, exercise

Effects of Depression

Series Health Issues

Relationship Trouble

Lack of productivity

Drug Dependency

“No one to speak and share feelings”

Differentiating Factors

	Accessibility	Accuracy	Service Offering
Heart Healer			AI/Professional
Wysa			AI
BetterHelp			Professional
MindCafe			Professional
ThoughtFull			Professional