Building Automated Testing and Bug Repair Infrastructure for IoT

Norihiro Yoshida, Ritsumeikan University, Japan

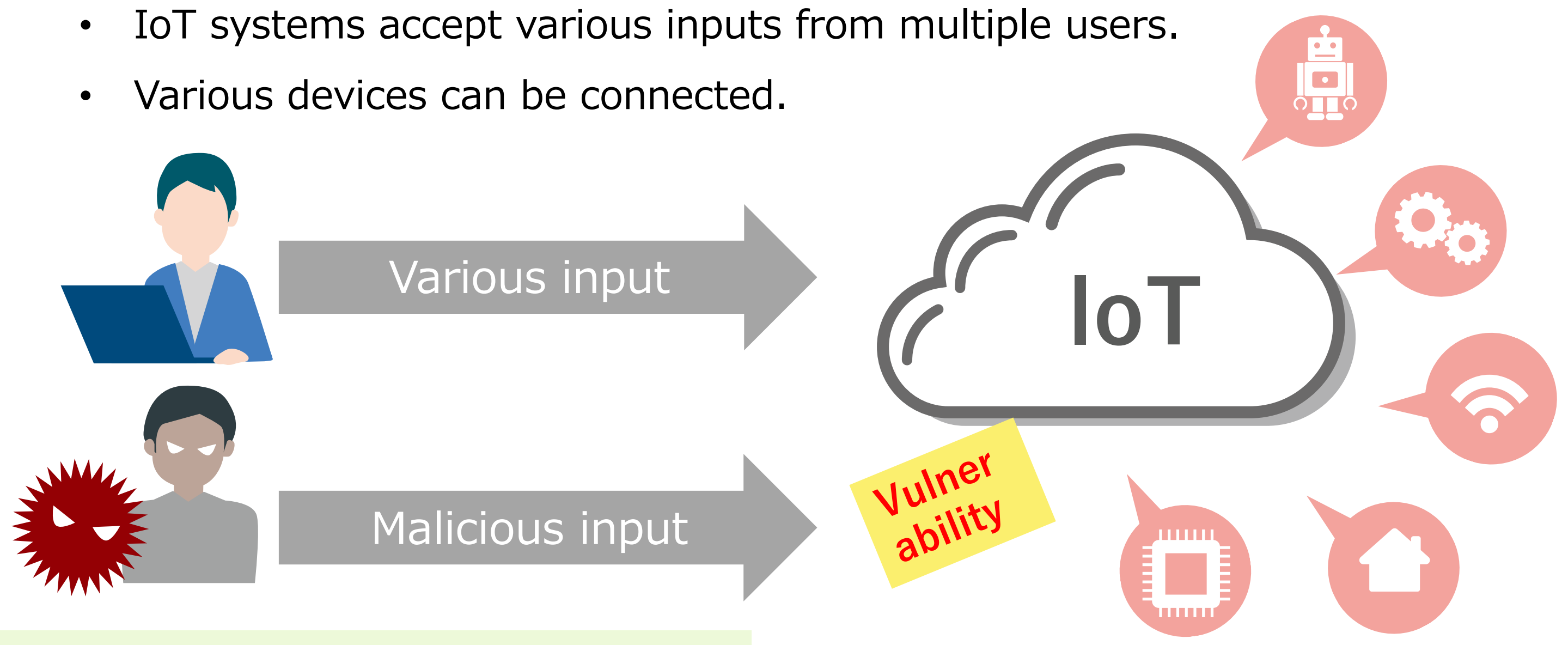
JST PRESTO [JPMJPR21PA]

RITSUMEIKAN

Testing and Debugging for IoT

Problem 1: Difficult to perform sufficient testing

- IoT systems accept various inputs from multiple users.
- Various devices can be connected.



Problem 2: Difficult to debug

- Vulnerabilities in IoT systems are complex [1]
- Difficult to identify and fix code with vulnerabilities

[1] A. Makhshari, A. Mesbah: IoT Bugs and Development Challenges, ICSE 2021

Research Goals

Problem1: Difficult to perform sufficient testing

→**Goal1:** Detection of vulnerabilities in IoT systems

Problem2: Difficult to debug

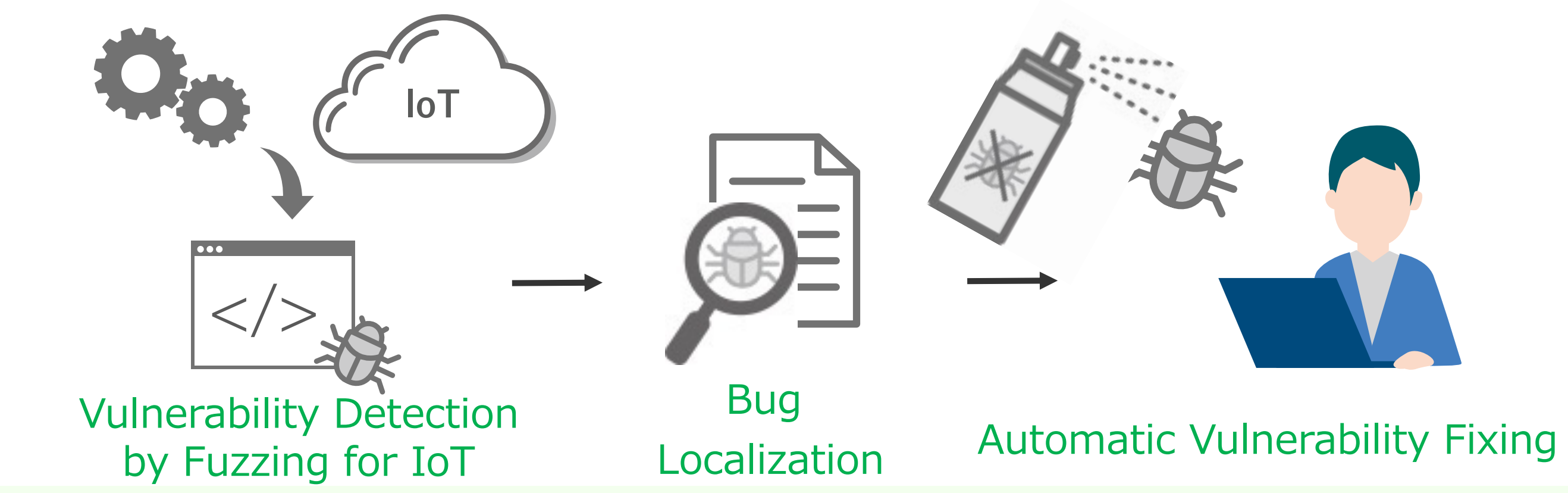
→**Goal2:** Seamless automation from vulnerability detection to repair

Development infrastructure
for automated vulnerability detection and repair

Automated Testing and Bug Repair

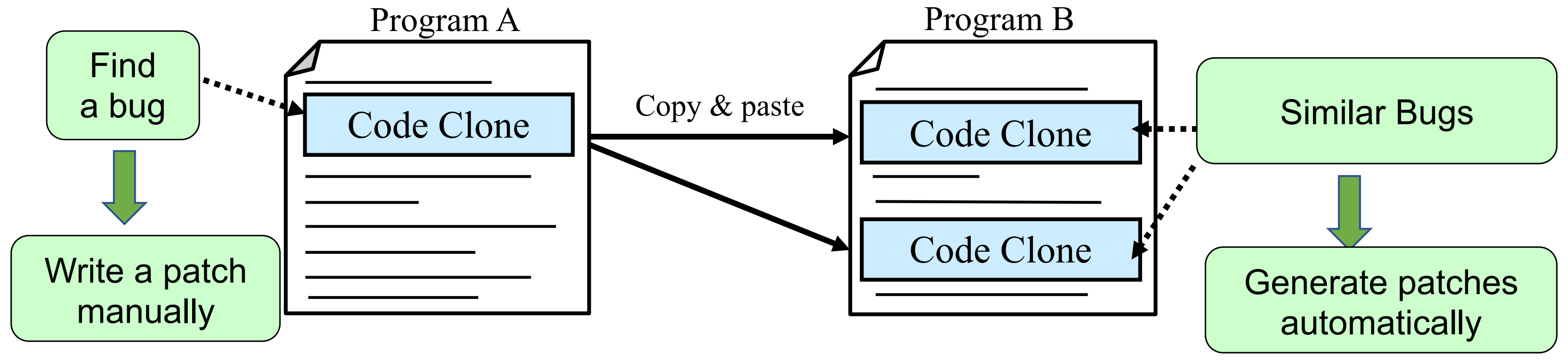
Fuzzing for IoT

Automated Bug Repair


IoT
&lt;/&gt;
Vulnerability Detection by Fuzzing for IoT
Bug Localization
Automatic Vulnerability Fixing

The steps from fuzzing to automated bug fixing are seamlessly automated by sandwiching Bug Localization between fuzzing and automated bug repair.

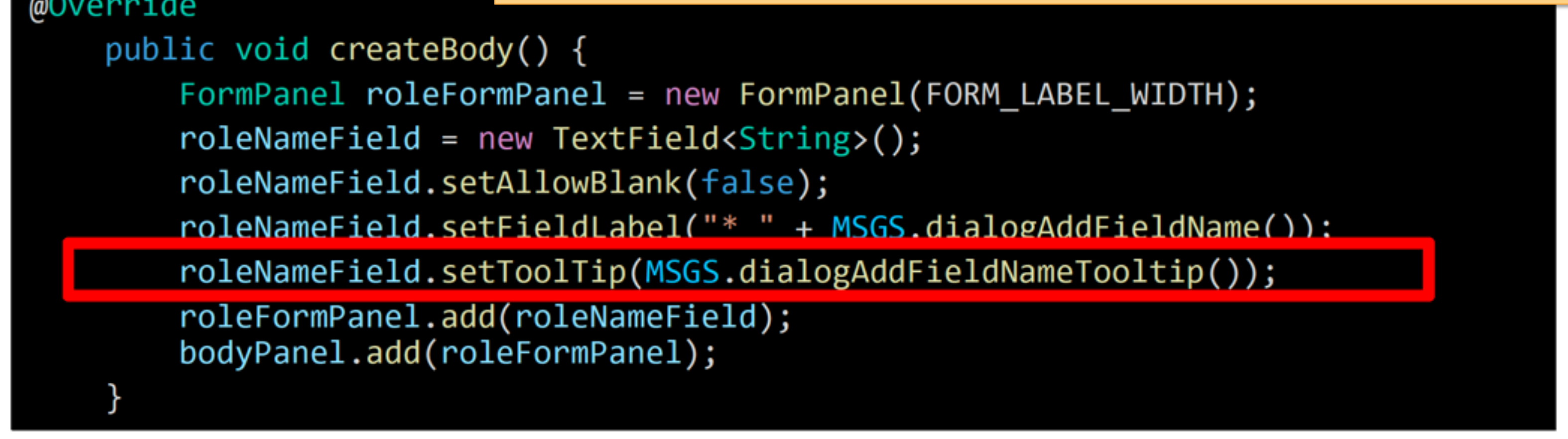
Patch Generation Based on Code Clone Detection



Automatically generates patches for code clones based on a single patch.

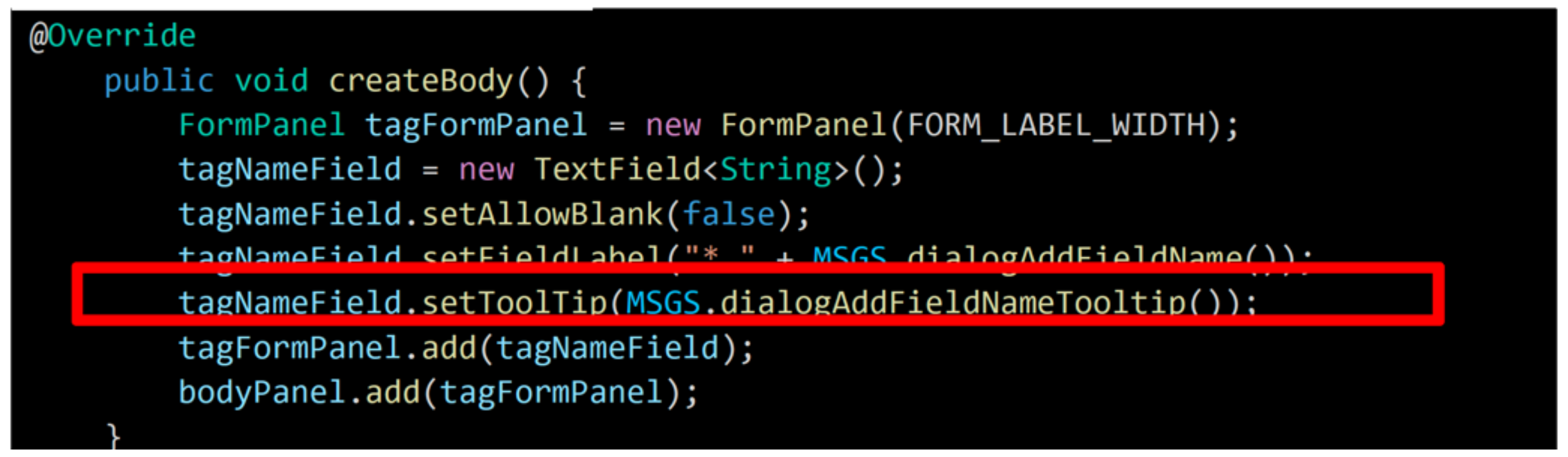
A patch to the upper side is generated based on the patch to the lower side.

RoleAddDialog.java



```
@Override
    public void createBody() {
        FormPanel roleFormPanel = new FormPanel(FORM_LABEL_WIDTH);
        roleNameField = new TextField<String>();
        roleNameField.setAllowBlank(false);
        roleNameField.setFieldLabel("* " + MSGS.dialogAddFieldName());
        roleNameField.setToolTip(MSGS.dialogAddFieldNameTooltip());
        roleFormPanel.add(roleNameField);
        bodyPanel.add(roleFormPanel);
    }
```

TagAddDialog.java



```
@Override
    public void createBody() {
        FormPanel tagFormPanel = new FormPanel(FORM_LABEL_WIDTH);
        tagNameField = new TextField<String>();
        tagNameField.setAllowBlank(false);
        tagNameField.setFieldLabel("* " + MSGS.dialogAddFieldName());
        tagNameField.setToolTip(MSGS.dialogAddFieldNameTooltip());
        tagFormPanel.add(tagNameField);
        bodyPanel.add(tagFormPanel);
    }
```

Patch generation based on code clone differences

Modifying the original patch based on the differences between code clones

Type-1: Exact match at the token level
→ Generate a patch to replace the token sequence as is

Type-2: Exact match except for variable names
→ Generate patches that do not modify unmatched variable names but replace other tokens

Type-3: Tokens have been added or deleted

e.g., adding arguments, conditional expressions

→ Generate patches to replace only the corresponding tokens (does not modify tokens that have been added or deleted)

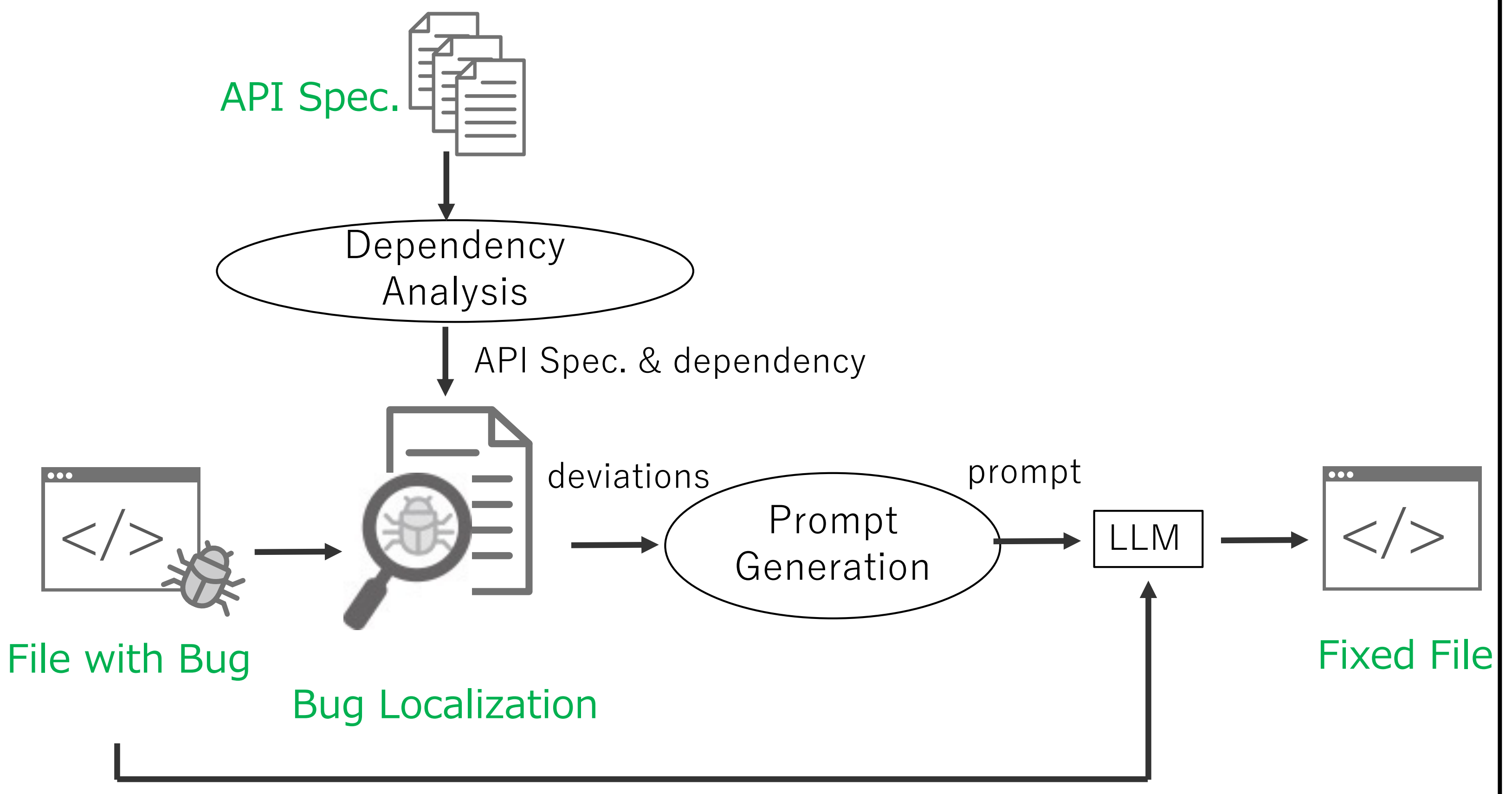
Twenty-six patches were successfully generated.

Automated Bug Repair Based on REST API specification

1. Detect deviations based on the REST API specification
2. Generate prompts from deviations and corresponding code fragments
3. Perform bug repair using LLM

Automated bug repair using Codex is better than the existing automated bug repair tools [2].

[2] Z.Fan, et al. Automated Repair of Programs from Large Language Models, ICSE 2023.



Preliminary Case Study

A preliminary case study was conducted for a program called the SwtichBot API.

a. Extract repositories using the SwitchBot API from GitHub
b. Extract hunks with modifications related to the REST API specification
c. Check if it can be repaired when entering the file before modification

- The proposed approach was able to repair 8/20 cases.
- Only 3/20 cases were repaired by using LLM alone.