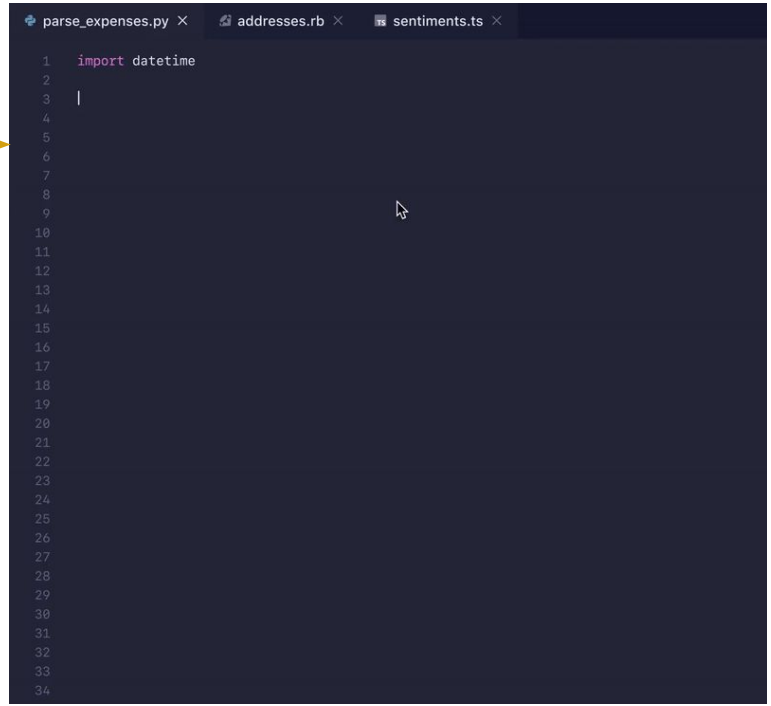
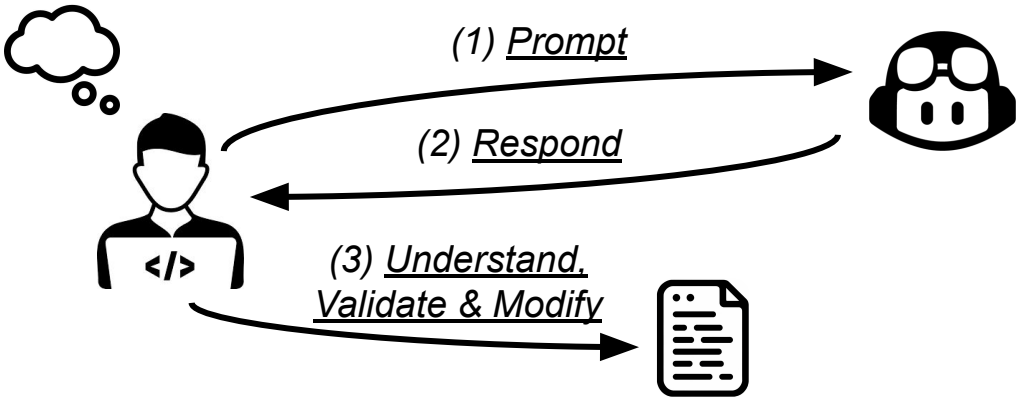
Towards Effective Validation and Integration of LLM-Generated Code

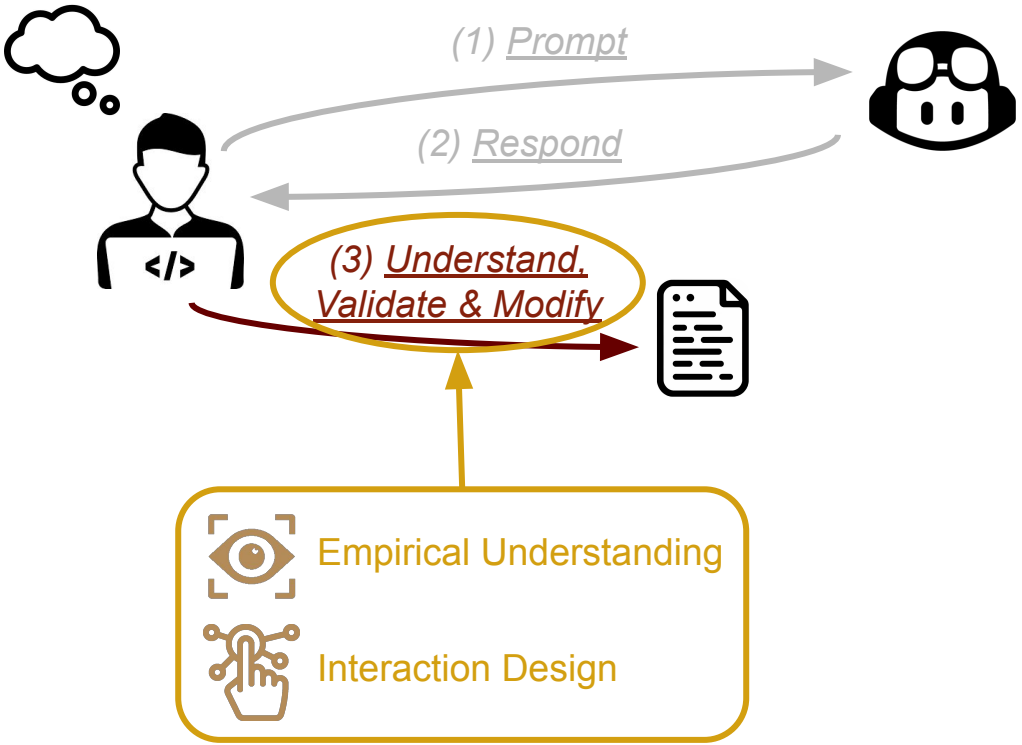
Ningzhi Tang
University of Notre Dame
VL/HCC 2024 Graduate Consortium

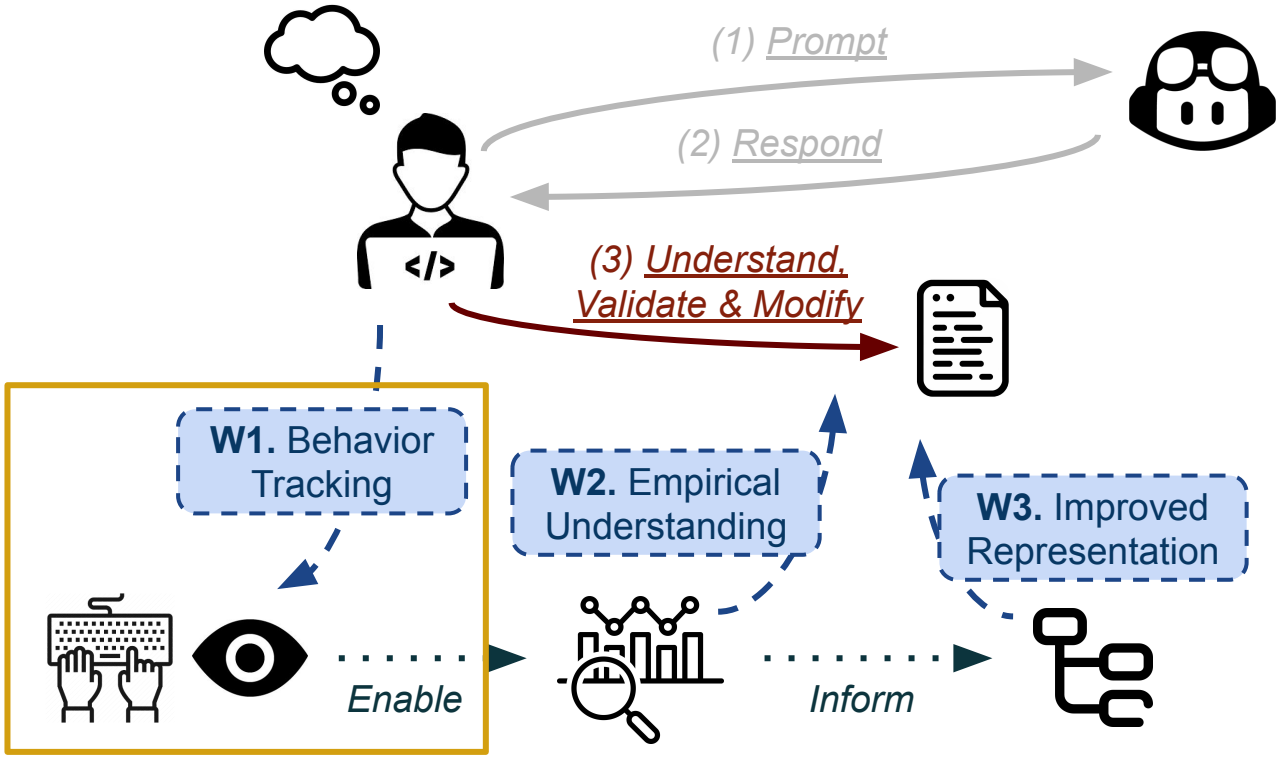
LLMs Transform Developer Workflows



** Source: GitHub Copilot, retrieved from https://github.com/features/copilot*


(1) Prompt
(2) Respond
(3) Understand, Validate & Modify


(1) Prompt
(2) Respond
(3) Understand, Validate & Modify
Empirical Understanding
Interaction Design



Ningzhi Tang *et al*. "CodeGRITS: A Research Toolkit for Developer Behavior and Eye Tracking in IDE". *ICSE-Demo 2024*.

CodeGRITS - Gaze Recording & IDE Tracking System

Subjective Analysis

Recall & Observer Biases

Video Recording

Survey

Interview

Complete With

Behavior Tracking

CodeGRITS: JetBrains Plugin

IDE Tracking

Interactions within IDE

Eye Tracking

Eye movements

CodeGRITS - Gaze Recording & IDE Tracking System

IDE Tracking

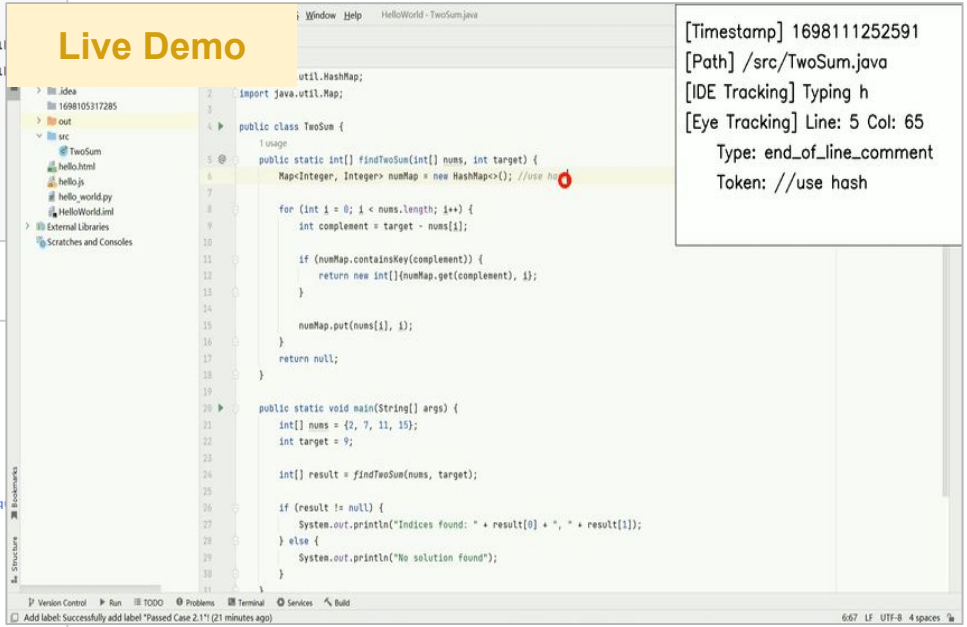
```
[OUTPUT_DIR]
├── [START_TIMESTAMP]
│   ├── ide_tracking.xml
│   ├── eye_tracking.xml
│   ├── archives
│   │   ├── [ARCHIVE_TIMESTAMP_1].a
│   │   ├── [ARCHIVE_TIMESTAMP_2].a
│   │   ├── ...
│   ├── screen_recording
│   │   ├── video_clip_1.mp4
│   │   ├── video_clip_2.mp4
│   │   ├── ...
│   │   ├── frames.csv
```

```
/Main.java" timestamp="
c/Main.java" timestamp="
t" path="/src/Main.java"
    <action id="GotoDeclaration" path="/src/Main.java" time
    <action id="Debug" path="/src/Main.java" timestamp="16
    <action id="NewClass" path="/src" timestamp="169621711
    <action id="RenameElement" path="/src/ABC.java" timesta
  </actions>
  <typings>
    <typing character="S" column="8" line="3" path="/src/M
    <typing character="y" column="9" line="3" path="/src/M
  </typings>
  <files>
    <file id="fileClosed" path="/src/Main.java" timestamp="
    <file id="selectionChanged" new_path="/src/ABC.java" o
        timestamp="1696216679330"/>
  </files>
  <mouses>
    <mouse id="mousePressed" path="/src/DEF.java" timestamp="1696217839651" x="642" y="120"/>
    <mouse id="mouseReleased" path="/src/DEF.java" timestamp="1696217840187" x="642" y="120"/>
  </mouses>
</ide_tracking>
```

Eye Tracking

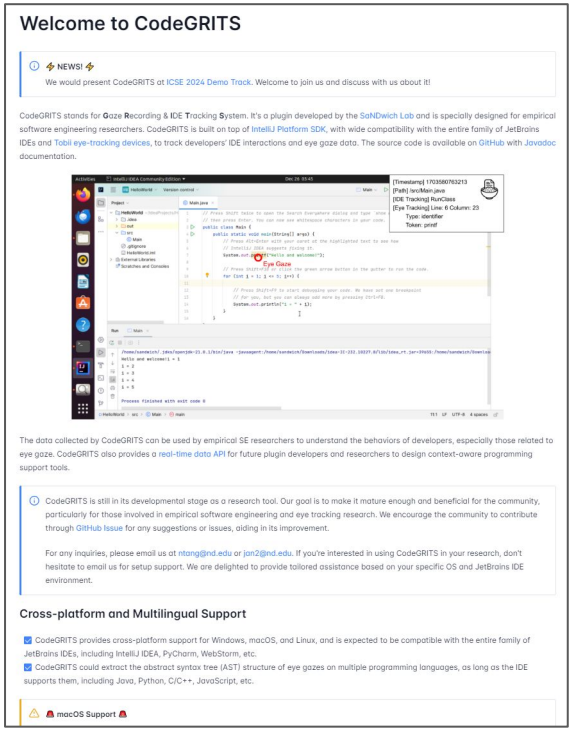
```
4166666666" gaze_point_y="0.17407407407407408" gaze_validity="1.0"
5662841796875" pupil_validity="1.0"/>
54166666666" gaze_point_y="0.17407407407407408" gaze_validity="1.0"
        pupil_diameter="2.718856811523437 5" pupil_validity="1.0"/>
    <location column="25" line="2" path="/src/Main.java" x="820" y="150"/>
    <ast_structure token="println" type="IDENTIFIER">
      <level end="2:26" start="2:19" tag="PsiIdentifier:println"/>
      <level end="2:26" start="2:8" tag="PsiReferenceExpression:System.out.println"/>
      <level end="2:42" start="2:8" tag="PsiMethodCallExpression:System.out.println(&quot;Hello world!&q
      <level end="2:43" start="2:8" tag="PsiExpressionStatement"/>
      <level end="3:5" start="1:43" tag="PsiCodeBlock"/>
      <level end="3:5" start="1:4" tag="PsiMethod:main"/>
      <level end="4:1" start="0:0" tag="PsiClass:Main"/>
    </ast_structure>
  </gaze>
</eye_tracking>
```

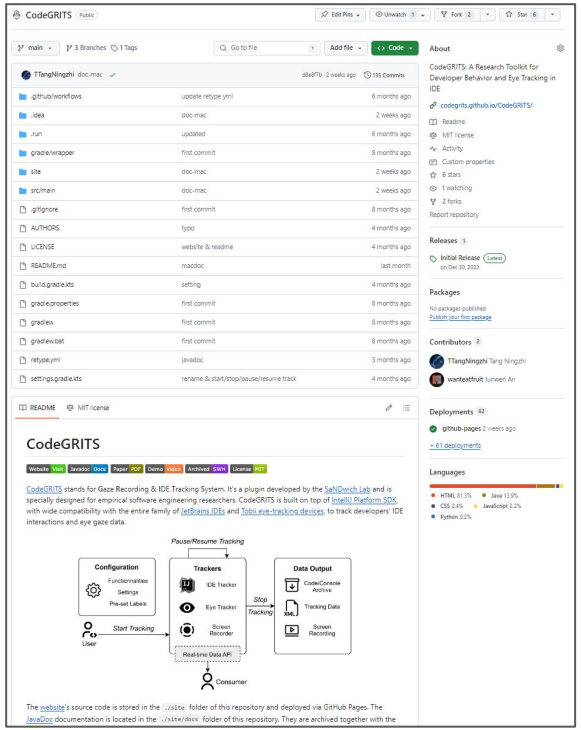
Live Demo



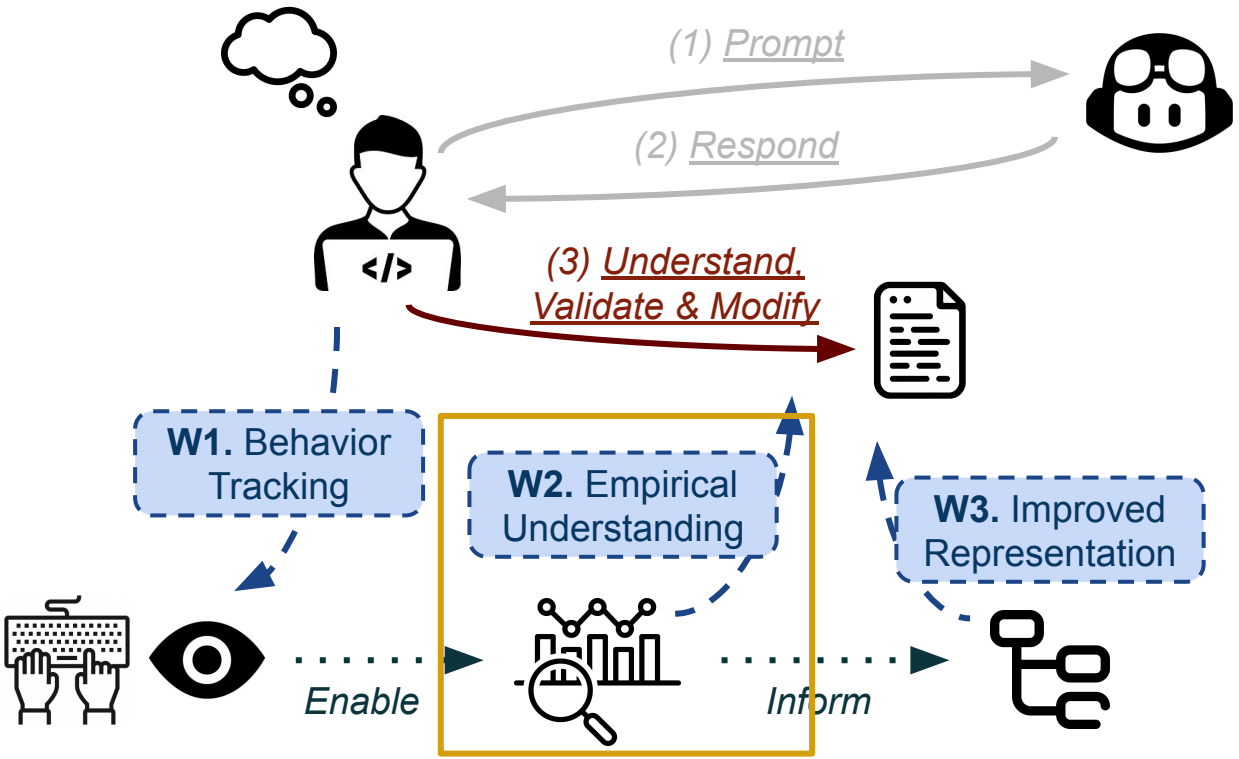
[Timestamp] 1698111252591
[Path] /src/TwoSum.java
[IDE Tracking] Typing h
[Eye Tracking] Line: 5 Col: 65
Type: end_of_line_comment
Token: //use hash

CodeGRITS - Gaze Recording & IDE Tracking System





codegrits.github.io/CodeGRITS



Ningzhi Tang *et al.* "Developer Behaviors in Validating and Repairing LLM-Generated Code Using IDE and Eye Tracking". *VL/HCC 2024*.

Research Question

RQ1. What are developers' perceptions and strategies that is specific for validating and repairing LLM-generated code?

RQ2. How does awareness of code provenance (i.e., whether the code is LLM-generated or human-written) affect code validation and repair behavior?

GitHub Copilot

🛠️ Specifically designed for coding

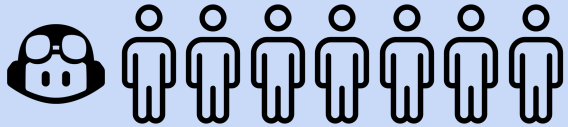
🌍 Widely used

📚 Previously studied in other works

Study Design

🏫 **University of Notre Dame**
28 participants

⏳ **Programming Experience**
5.5 years (average)



Informed Group (14)

Non-Informed Group (14)

🔧 **Validate and Repair Copilot-Generated Code**

- Algorithm Design
- GUI Programming
- Object-Oriented Programming

Study Result

LLMs make <u>distinct types of mistakes</u> that are uncommon for human developers.

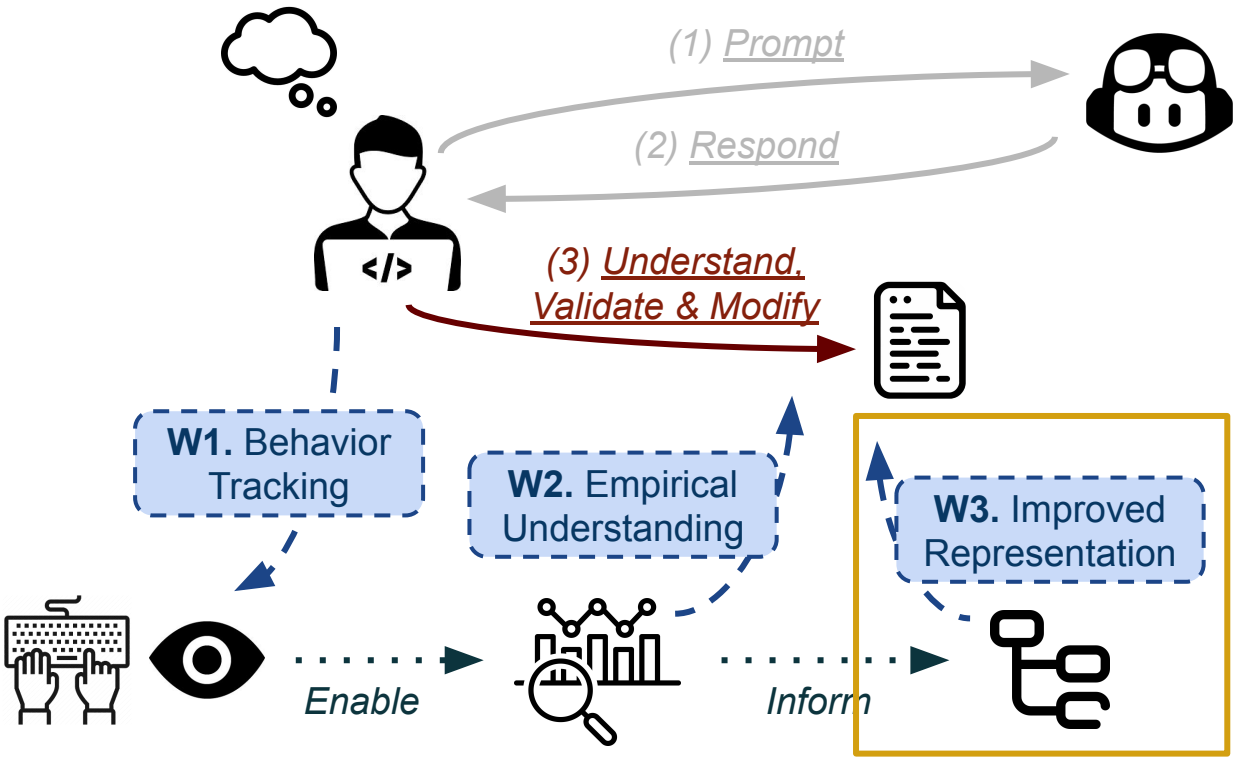
"[...] tend to hard-code test samples and generate hallucinated objects." (P21)

Developers like to use LLMs to <u>generate inline comments</u> for understanding.

Developers display a <u>high switching workload</u> between code and prompts.

When uninformed, about 80% of developers <u>cannot distinguish code provenance</u>, and they <u>fixed fewer bugs</u> and <u>showed different behaviors</u>.

💡 Adaptive systems are designed based on the unique characteristics of LLM-generated code, as well as improved awareness of code provenance.



"Tree Representations Enabled Multi-Level and Syntax-Aware LLM Code Interactions". *In Progress*.

Developers Using LLMs to Understand and Modify LLM-Generated Code

Particularly Those Unfamiliar With Current Technology

Current Interaction Paradigm: Textual Representation

```java
public static int binarySearch(int[] arr, int element) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == element) return mid;
        if (arr[mid] < element) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Mouse Selection + Chatting

```java
public static int binarySearch(int[] arr, int element) {
    int left = 0, right = arr.length - 1;
    // Loop until the left and right pointers meet
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == element) return mid;
        if (arr[mid] < element) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Inline Comment/Code Generation

Developers Using LLMs to Understand and Modify LLM-Generated Code

Particularly Those Unfamiliar With Current Technology

Current Interaction Paradigm: Textual Representation

It works well for intuitive and flexible use... But are there any limitations?

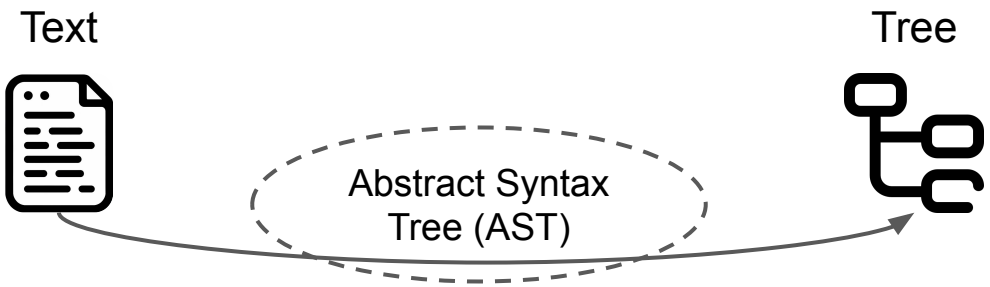
(1) Unsupporting **Multi-Level Interaction**

It mainly suggests local edits, but developers need to understand and modify code at different levels of abstraction, from local statements to overall functionality.

(2) Unawareness of **Code Syntax**

It inhibits the design of many syntax-aware interactions, and may reduce the accuracy and context-awareness needed to support developers effectively.

New Representations to Support Multi-Level and Syntax-Aware Code Interactions



Tree Representation of Code Built from the Abstract Syntax Tree (AST)

- 1 Method:binarySearch
  - 1.1 Declaration
  - 1.2 While
    - 1.2.1 Declaration
    - 1.2.2 If
    - 1.2.3 If
  - 1.3 Return

Interaction Features to Support LLM Code Understanding & Modification

Example Feature: Multi-Level Code Explanations

Understand the code at <u>different levels of detail</u> with less effort.

```java
// Returns the element's index if present in the array; otherwise, returns -1
public static int binarySearch(int[] arr, int element) {
    int left = 0, right = array.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == element) return mid;
        if (arr[mid] < element) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Explain Node

A

- 1 Method:binarySearch
  - 1.1 Declaration
  - 1.2 While
    - 1.2.1 Declaration
    - 1.2.2 If
    - 1.2.3 If
  - 1.3 Return

B

Compares the middle element with the target and updates the left and right pointers.

C

Example Feature: Procedurally Prompted Editing

```
// Returns the element's index if present in the array; otherwise, returns -1
public static int binarySearch(int[] arr, int element) {
    int left = 0, right = array.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == element) return mid;
        if (arr[mid] < element) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

A

Explain Node

- 1 Method:binarySearch
  - 1.1 Declaration
  - 1.2 While
    - 1.2.1 Declaration
    - 1.2.2 If
    - 1.2.3 If
  - 1.3 Return

B

Compares the middle element with the target and updates the left and right pointers.

C

[~~Compares~~] Tests whether the middle element [with] is the left-most target and updates the left and right pointers.

D

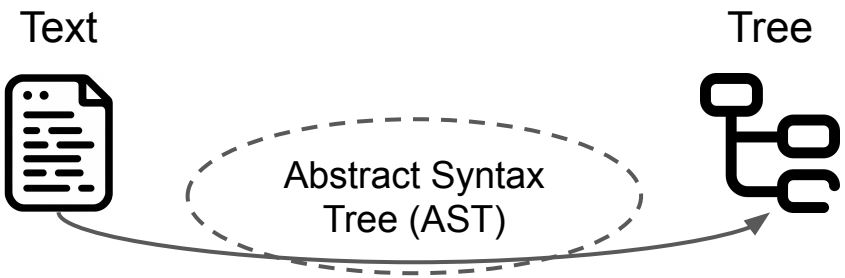
Modify Explanations

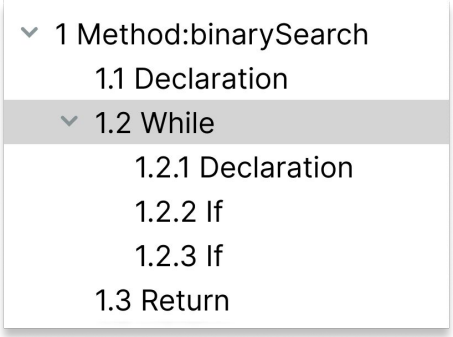
Prompt LLM to Modify Code

```
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == element) {
            while (mid > 0 && arr[mid - 1] == element) {
                mid--;
            }
            return mid;
```

E

Tree Representations for Multi-Level and Syntax-Aware Code Interactions

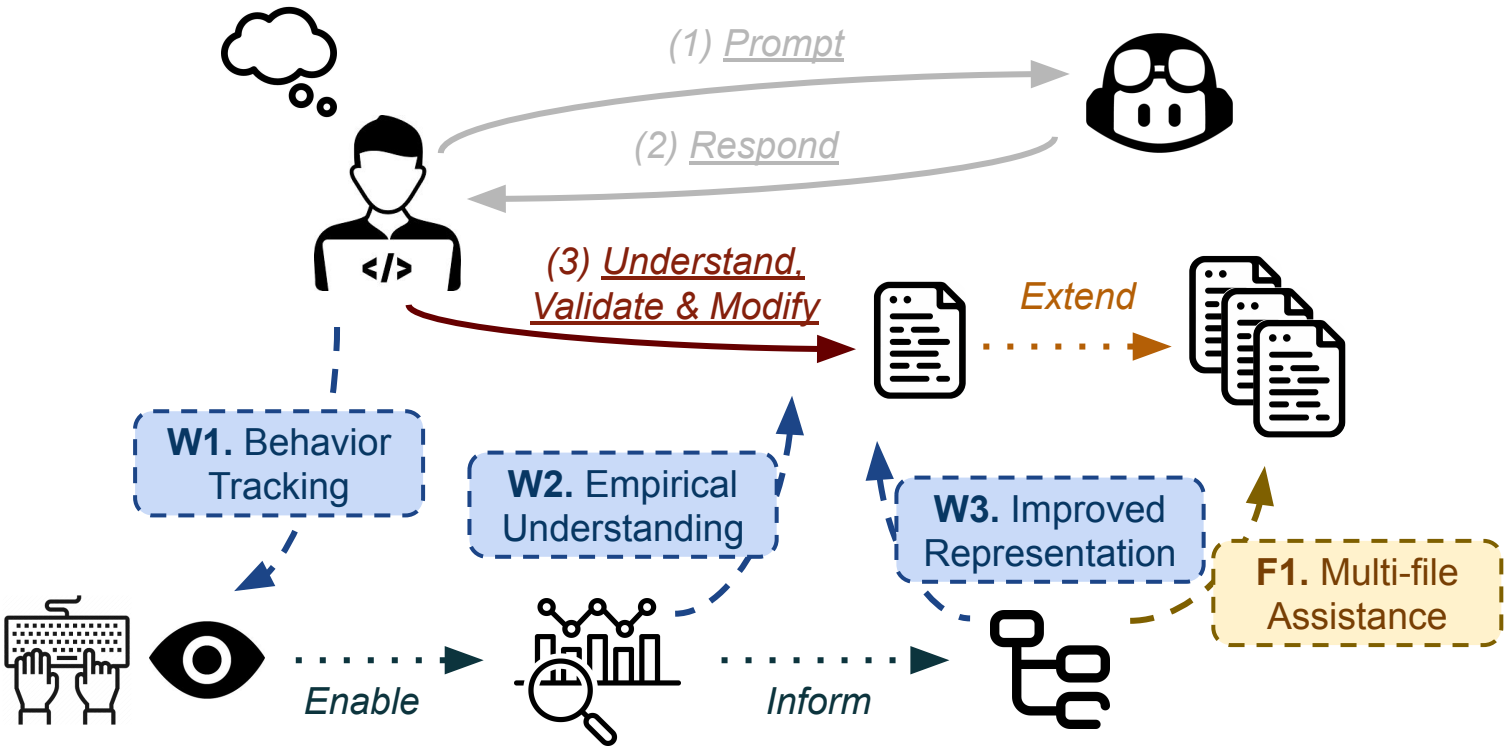




Feature 1. Multi-level Code Explanations

Feature 2. Procedurally Prompted Editing

More …?



"Tree Representation Based Interaction for Multi-File Code Validation and Modification"

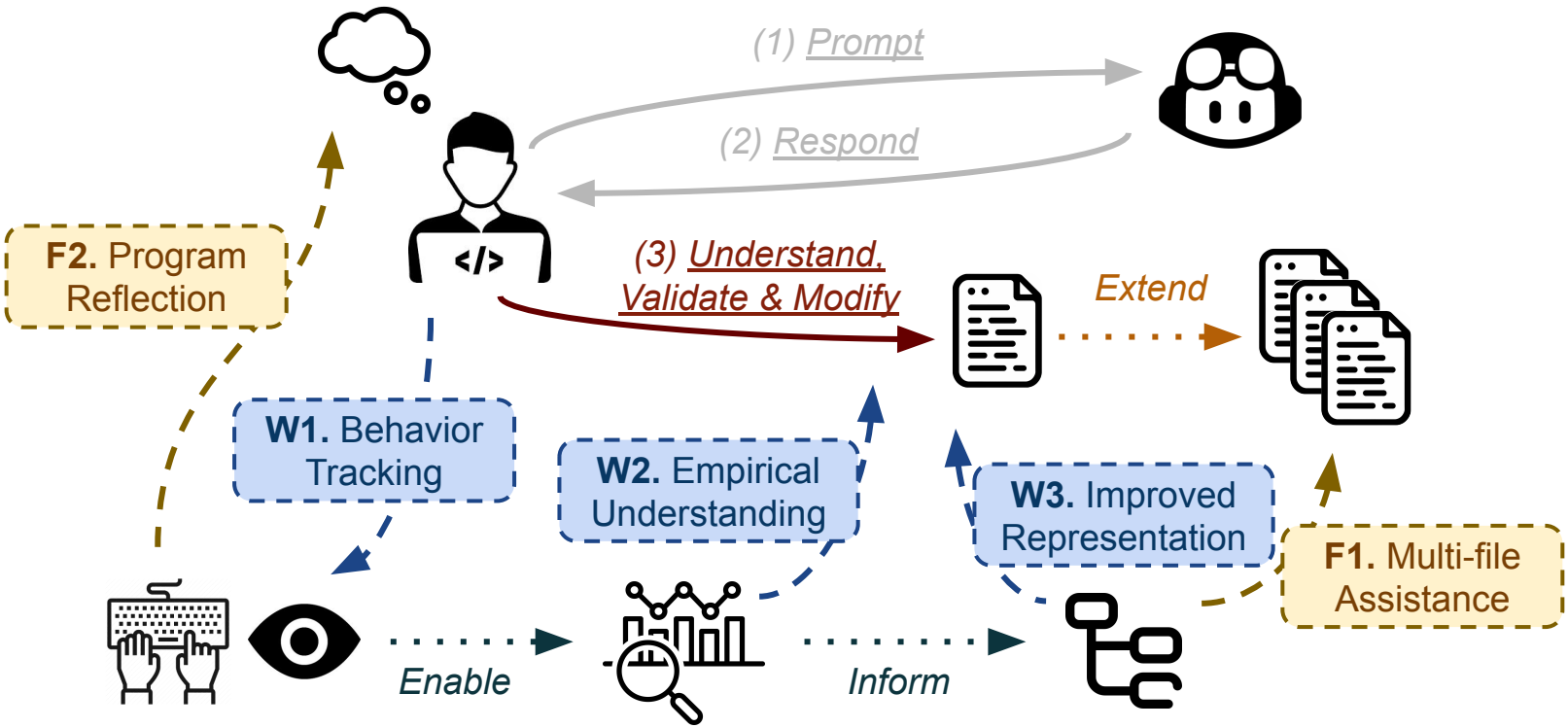
Future Work 1. Extending Tree Representation to Support Multiple Files

What will happen when LLM generate code across multiple files?

(1) LLM edits should be validated individually to address static and runtime errors.

(2) Categorizing and aggregating them at different abstraction levels aid validation.

💡 Multi-level and syntax-aware interactions may still be valuable for multi-file scenarios.



"Program Reflection Through Developer Behavior Tracking and LLM Insights Generation"

Future Work 2. Enabling Program Reflection Through Behavior Tracking

Self-reflection enhances developers' ability to evaluate and improve their programming skills.

IDE Interactions & Eye Tracking (CodeGRITS) → Programming Practice Understanding & Suggestions

Low Level Behaviors

High-Level Insights

Observation: *Frequently jumps back and forth between multiple files.*

Suggestion: *Use IDE features like bookmarks or call hierarchy for more efficient navigation.*

🛠️ **Method Gap:** How to model the transition from low-level behaviors to high-level insights?

🧩 **Interface Gap:** How to effectively convey data-driven insights to human developers?

Acknowledgment

This project was support in part by NSF grants CCF-2211428 and CCF-2100035. Any opinions, findings, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

Towards Effective Validation and Integration of LLM-Generated Code

Ningzhi Tang, University of Notre Dame

Ningzhi Tang

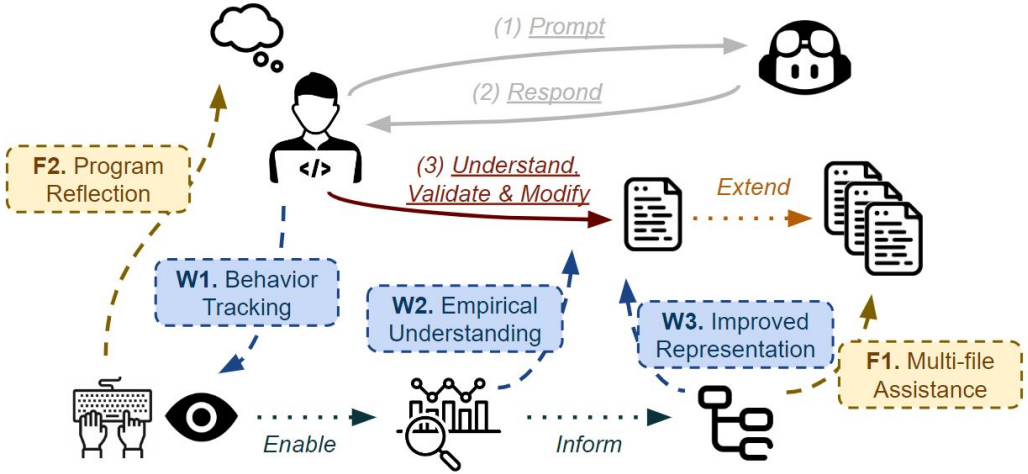
nztang.com

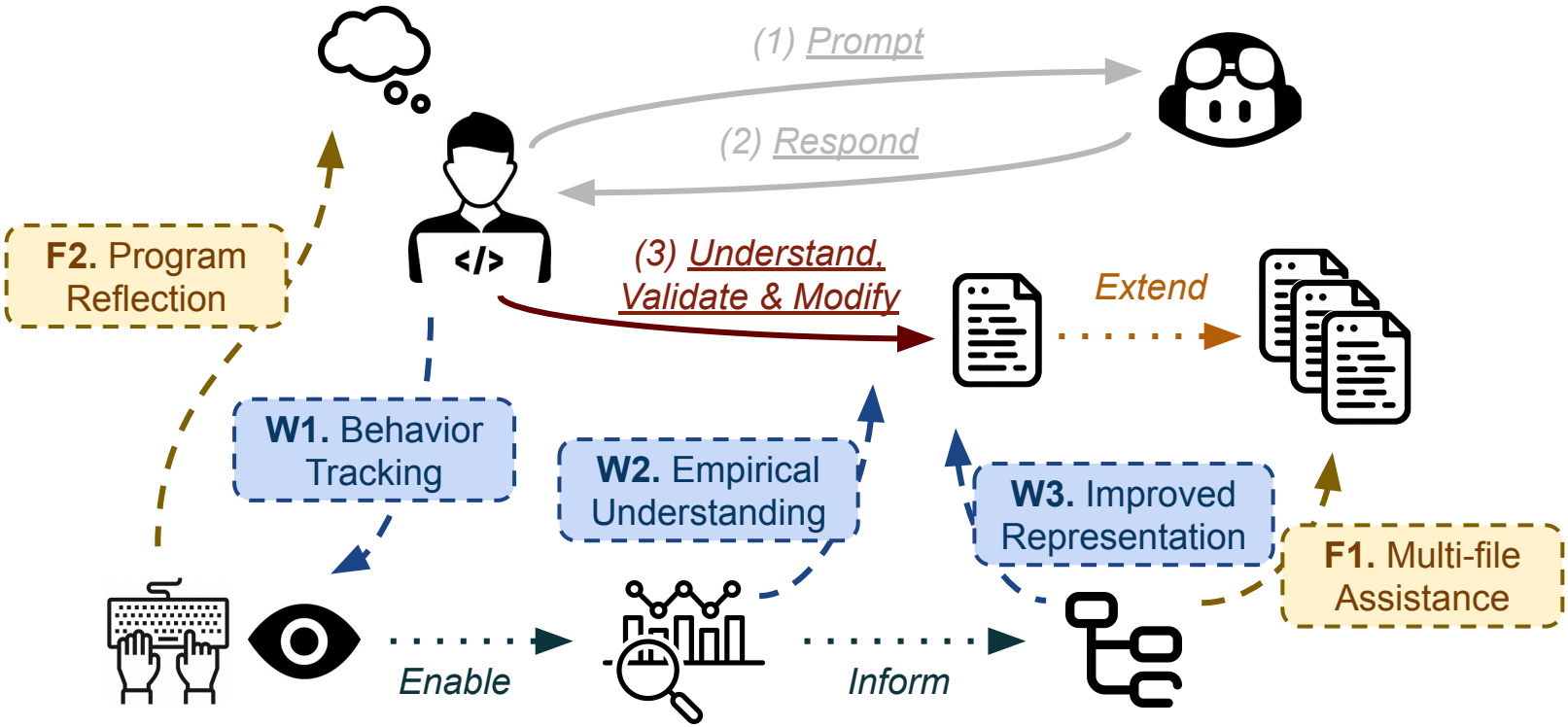
ntang@nd.edu

@TangNingzhi

ningzhi_tang

codegrits.github.io/
CodeGRITS




(1) Prompt
(2) Respond
(3) Understand, Validate & Modify
Extend
F2. Program Reflection
W1. Behavior Tracking
W2. Empirical Understanding
W3. Improved Representation
F1. Multi-file Assistance
Enable
Inform