



UNIVERSITY OF
NOTRE DAME



Towards Effective Validation and Integration of LLM-Generated Code

Ningzhi Tang

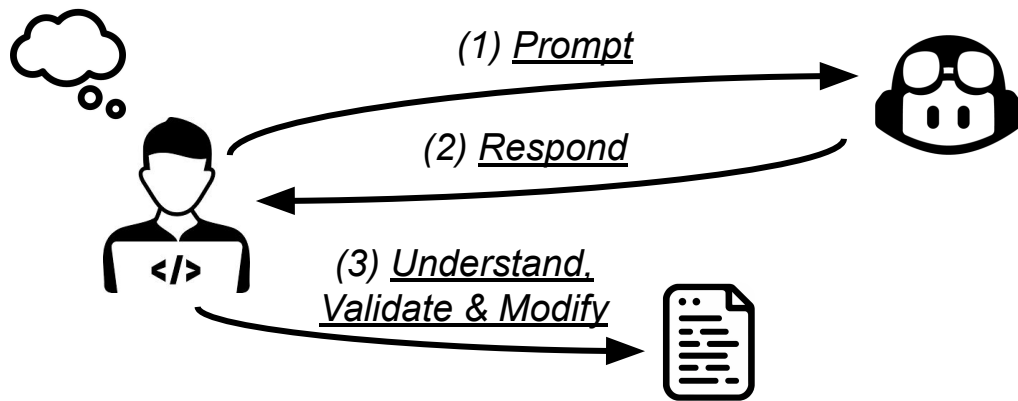
University of Notre Dame

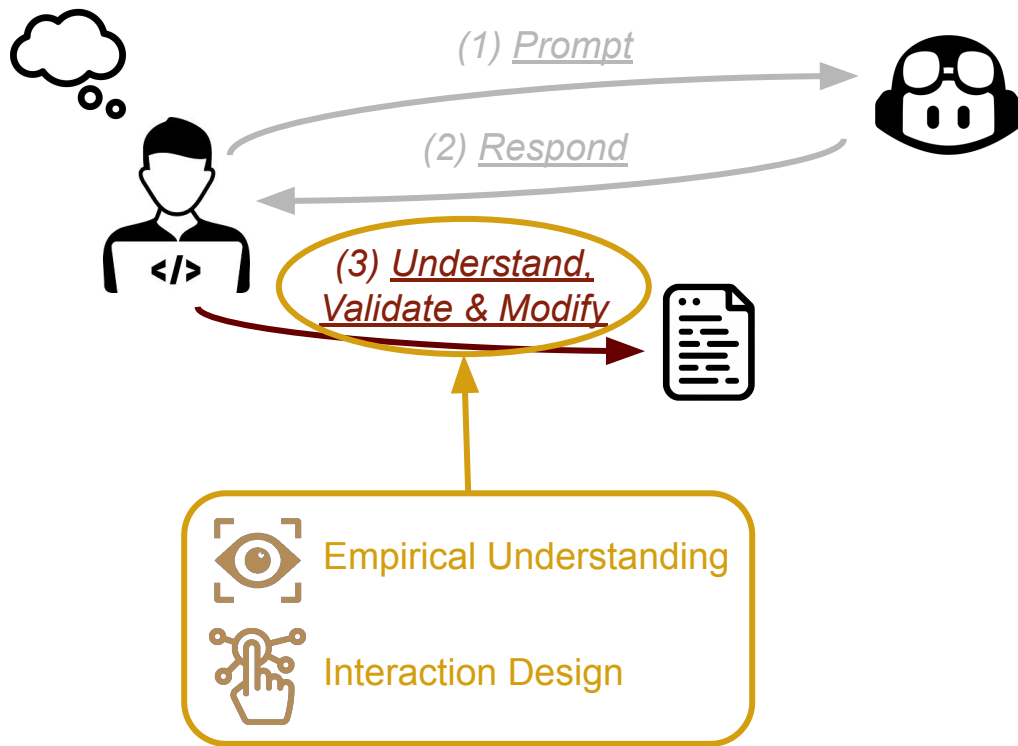
VL/HCC 2024 Graduate Consortium

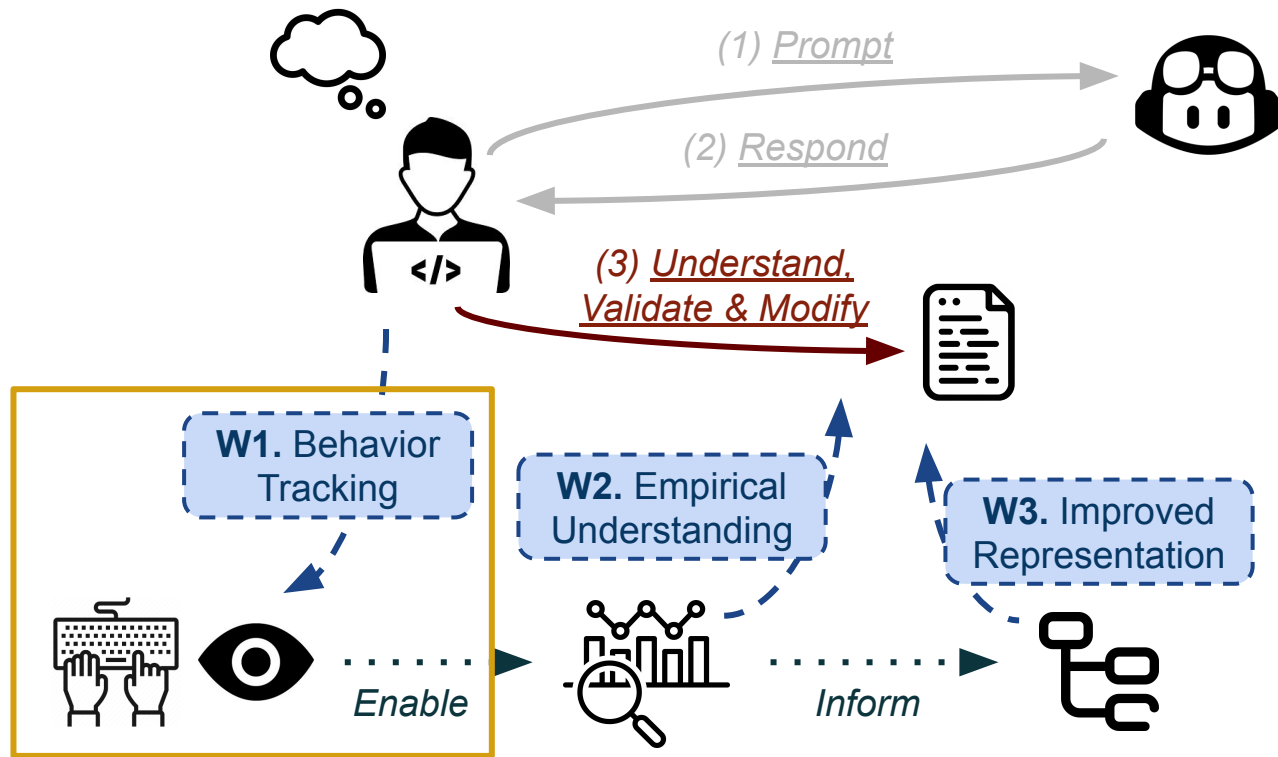
LLMs Transform Developer Workflows



```
parse_expenses.py × addresses.rb × sentiments.ts ×  
1 import datetime  
2  
3 |  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34
```







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

```

/Main.java timestamp="1696216679330"
c/Main.java timestamp="1696216679330"
t path="/src/Main.java" timestamp="1696216679330"
<action id="GoToDeclaration" path="/src/Main.java" timestamp="1696216679330">
</action>
<action id="Debug" path="/src/Main.java" timestamp="1696216679330">
</action>
<action id="NewClass" path="/src" timestamp="1696217111111">
</action>
<action id="RenameElement" path="/src/ABC.java" timestamp="1696216679330">
</action>
</actions>
<typings>
<typing character="S" column="8" line="3" path="/src/Main.java" timestamp="1696216679330">
</typing>
<typing character="y" column="9" line="3" path="/src/Main.java" timestamp="1696216679330">
</typing>
</typings>
</files>
<file id="fileClosed" path="/src/Main.java" timestamp="1696216679330">
</file>
<file id="selectionChanged" new_path="/src/ABC.java" old_path="/src/Main.java" timestamp="1696216679330">
</file>
</files>
<mouses>
<mouse id="mousePressed" path="/src/DEF.java" timestamp="1696217839651" x="642" y="120">
</mouse>
<mouse id="mouseReleased" path="/src/DEF.java" timestamp="1696217840187" x="642" y="120">
</mouse>
</mouses>
</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

```

Eye Tracking

```

5416666666666 gaze_point_y="0.17407407407408" gaze_validity="1.0"
3662841796875 pupil_validity="1.0"/>
5416666666666 gaze_point_y="0.17407407407408" gaze_validity="1.0"
pupil_diameter="2.7188568115234375" 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!&quot;)/>
<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

Welcome to CodeGRITS

NEWS! We would present CodeGRITS at ICSE 2024 Demo Track. Welcome to join us and discuss with us about it!

CodeGRITS stands for Gaze Recording & IDE Tracking System. It's a plugin developed by the [SciNDeWich Lab](#) and is specially designed for empirical software engineering researchers. CodeGRITS is built on top of [IntelliJ Platform SDK](#), with wide compatibility with the entire family of JetBrains IDEs and [Tobii eye-tracking devices](#), to track developers' IDE interactions and eye gaze data. The source code is available on GitHub with [JavaDoc](#) documentation.



The data collected by CodeGRITS can be used by empirical SE researchers to understand the behaviors of developers, especially those related to eye gaze. CodeGRITS also provides a [real-time data API](#) for future plugin developers and researchers to design context-aware programming support tools.

CodeGRITS is still in its developmental stage as a research tool. Our goal is to make it mature enough and beneficial for the community, particularly for those involved in empirical software engineering and eye tracking research. We encourage the community to contribute through [GitHub Issue](#) for any suggestions or issues, aiding in its improvement.

For any inquiries, please email us at ntang@ind.edu or jan2@ind.edu. If you're interested in using CodeGRITS in your research, don't hesitate to email us for setup support. We are delighted to provide tailored assistance based on your specific OS and JetBrains IDE environment.

Cross-platform and Multilingual Support

- CodeGRITS provides cross-platform support for Windows, macOS, and Linux, and is expected to be compatible with the entire family of JetBrains IDEs, including IntelliJ IDEA, PyCharm, WebStorm, etc.
- CodeGRITS could extract the abstract syntax tree (AST) structure of eye gazes on multiple programming languages, as long as the IDE supports them, including Java, Python, C/C++, JavaScript, etc.

macOS Support

CodeGRITS

main 3 Branches 1 Tags

Go to file Add file Code

About CodeGRITS: A Research Toolkit for Developer Behavior and Eye Tracking in IDE

codegrits.github.io/CodeGRITS/

MIT license Activity Custom properties 6 stars 1 watching 2 forks Report repository

Releases 1 Initial Release on Dec 30, 2023

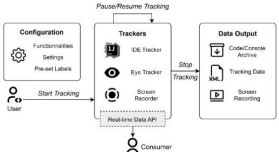
Packages No packages published Publish your first package

Contributors 2 TjiangNinghi Tjiang Ninghi Wentaozhao Junwen An

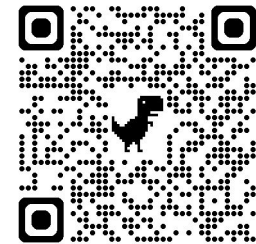
Deployments 1 github-pages 2 weeks ago E1 deployments

Languages HTML 81.5% Java 13.5% CSS 2.6% JavaScript 2.2% Python 2.2%

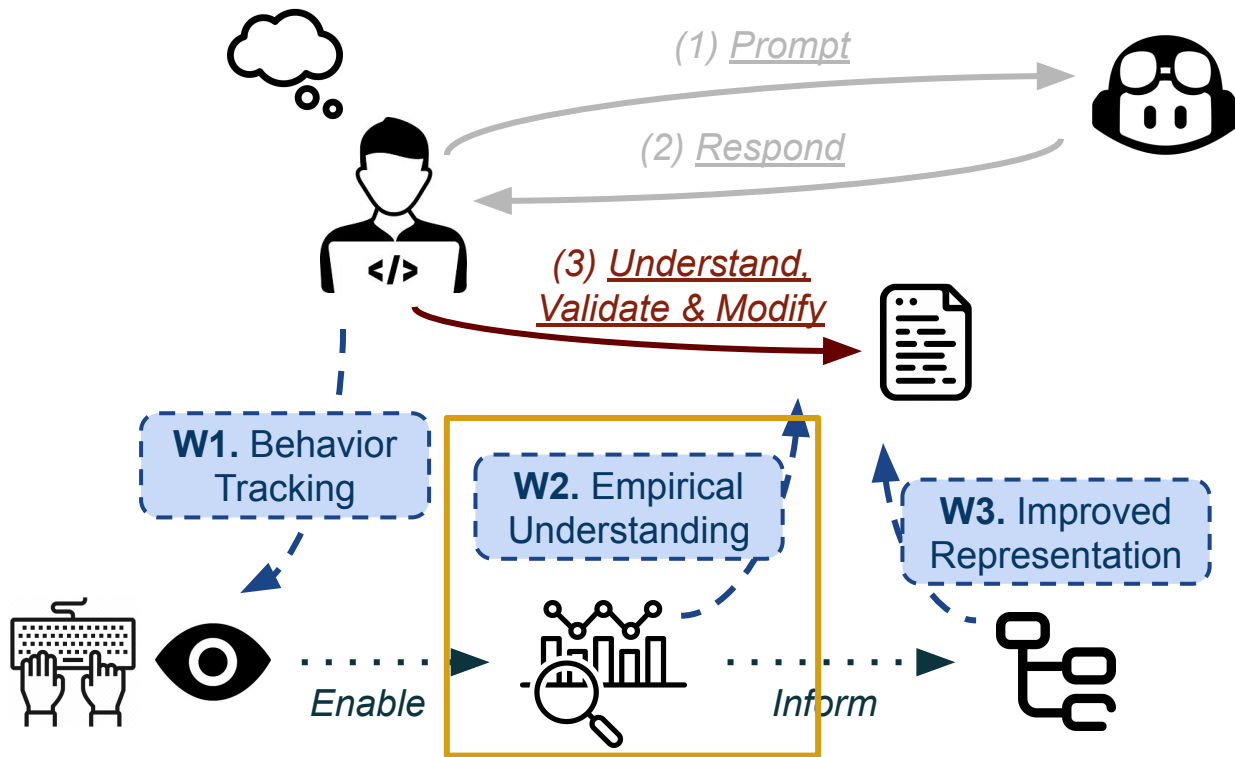
CodeGRITS stands for Gaze Recording & IDE Tracking System. It's a plugin developed by the [SciNDeWich Lab](#) and is specially designed for empirical software engineering researchers. CodeGRITS is built on top of [IntelliJ Platform SDK](#) with wide compatibility with the entire family of [JetBrains IDEs](#) and [Tobii eye-tracking devices](#), to track developers' IDE interactions and eye gaze data.



The `utils` source code is stored in the `utils` folder of this repository and deployed via GitHub Pages. The `JavaDoc` documentation is located in the `utils/html/docs` folder of this repository. They are archived together with the



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?





 Specifically designed for coding

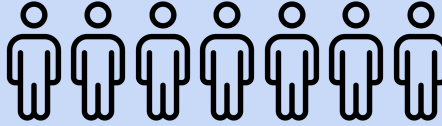
 Widely used

 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 distinct types of mistakes that are uncommon for human developers.

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

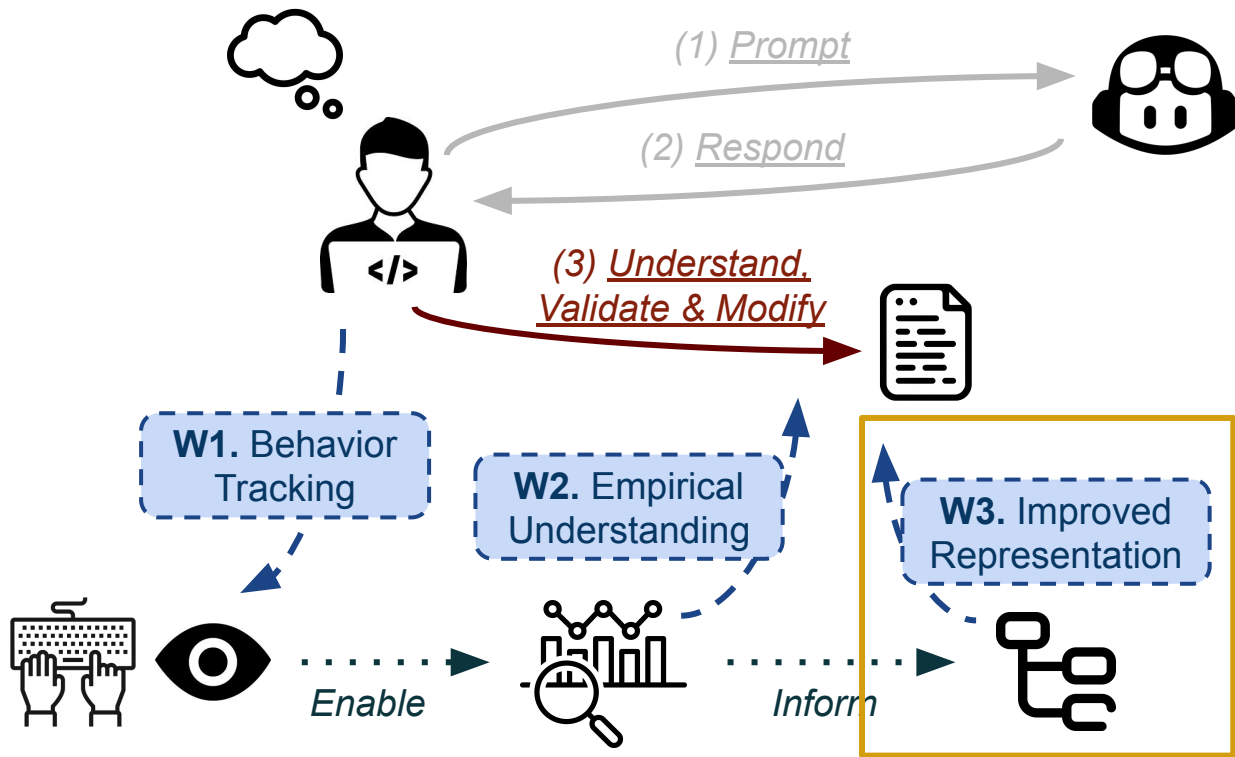
Developers like to use LLMs to generate inline comments for understanding.

Developers display a high switching workload between code and prompts.

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



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**

```
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

```
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**?*

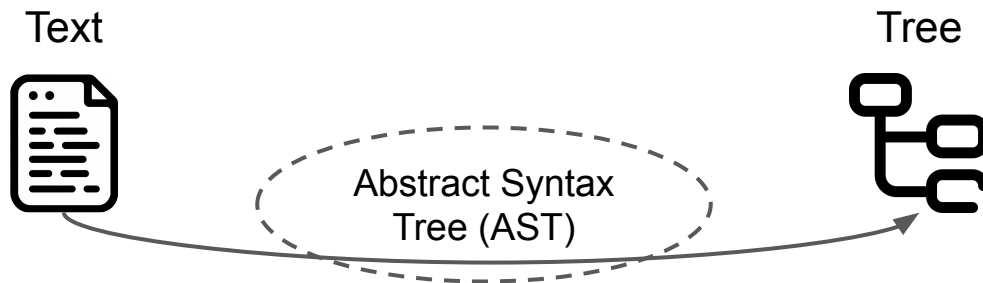
(1) Unsupporting Multi-Level Interaction

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 different levels of detail with less effort.

```
// 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 = 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;
}
```

Explain Node A

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

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;
}
```

Explain Node A

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

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

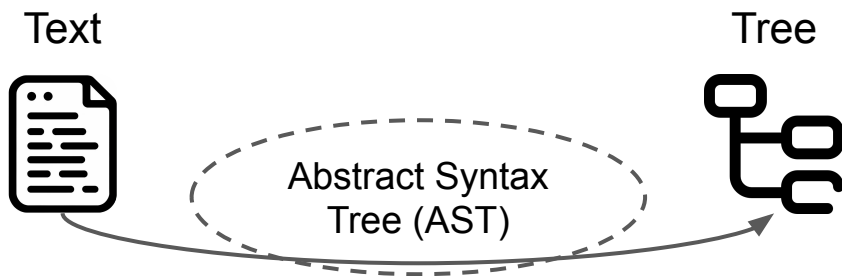
```
while (left <= right) {
    int mid = left + (right - left) / 2;
    << if (arr[mid] == element) {
        while (mid > 0 && arr[mid - 1] == element) {
            mid--;
        }
    }
    return mid;
}
```

E

Modify Explanations

Prompt LLM to Modify Code

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

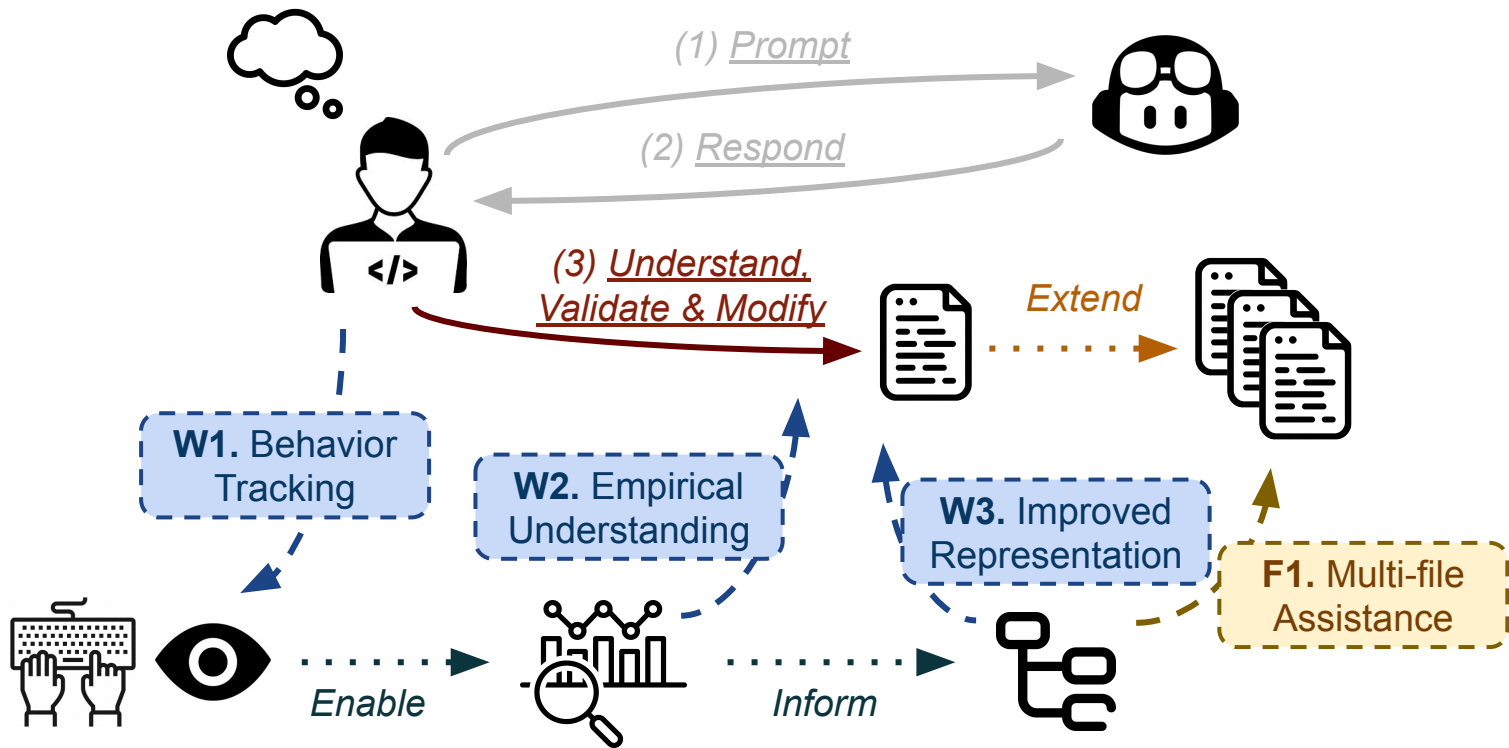


```
▼ 1 Method:binarySearch
  1.1 Declaration
  ▼ 1.2 While
    1.2.1 Declaration
    1.2.2 If
    1.2.3 If
  1.3 Return
```

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

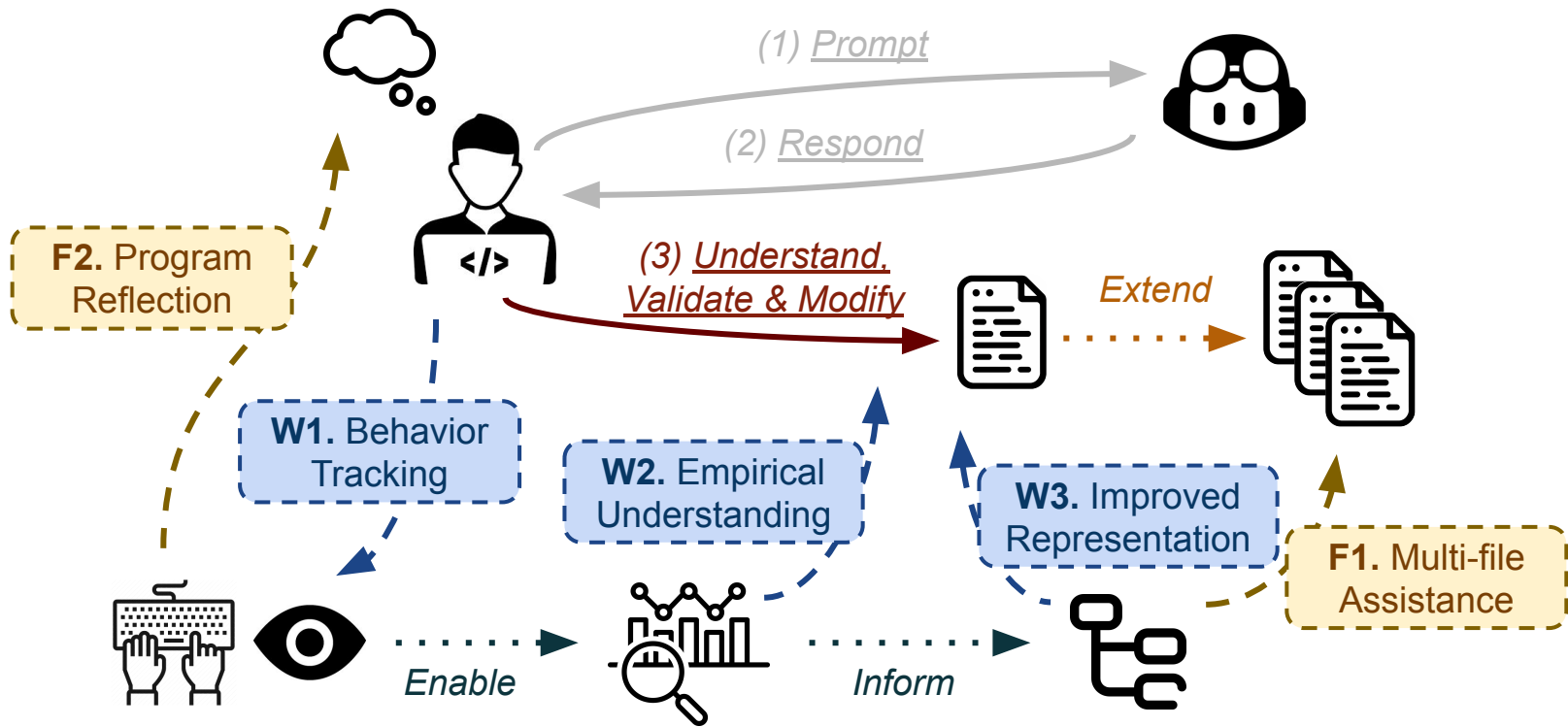
What will happen when LLM generate code across multiple files?



- (1) LLM edits should be validated individually to address static and runtime errors.
- (2) Categorizing and aggregating them at different abstraction levels aid validation.



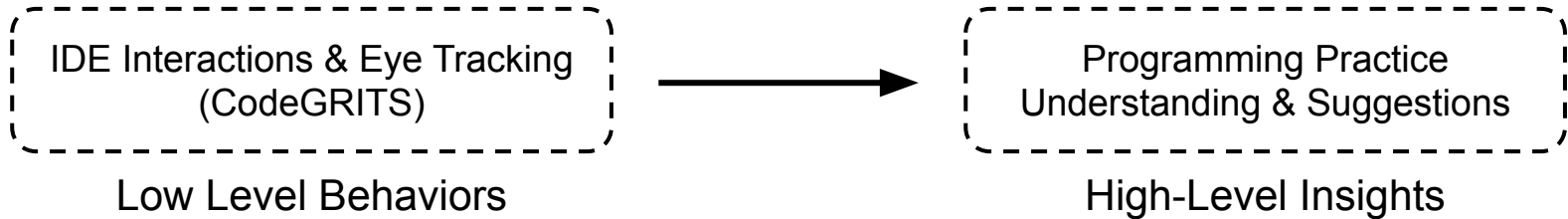
Multi-level and syntax-aware interactions may still be valuable for multi-file scenarios.



“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.



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

