



# Developer Behaviors in Validating and Repairing LLM-Generated Code Using IDE and Eye Tracking

Ningzhi Tang\*, Meng Chen\*, Zheng Ning, Aakash Bansal,  
Yu Huang, Collin McMillan, Toby Jia-Jun Li

***VL/HCC 2024***

# 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
```

# LLMs Make Distinct Types of Mistakes

```
Lion(String name, double maneSize, String colorOfMane, String lionType){  
    super(name);  
    this.maneSize = maneSize;  
    this.colorOfMane = colorOfMane;  
    this.lionType = lionType;  
    this.setSpecies("Lion");  
    this.setConservationStatus("Not Extinct");  
    this.setEatingHabits("Carnivores");  
    this.setFeatherColor("All Colors");  
    this.setBeakShape("Short and Conical");  
    this.setEggSize(25);  
}
```

**Bird or Lion?**





```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and return the list of triples (date, amount, currency)
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime (date, "%Y-%m-%d"),
                        float (value),
                        currency))
    return expenses

expenses_data = '''2023-01-02 -34.01 USD
2023-01-03 2.59 DKK
2023-01-03 -2.72 EUR'''
```

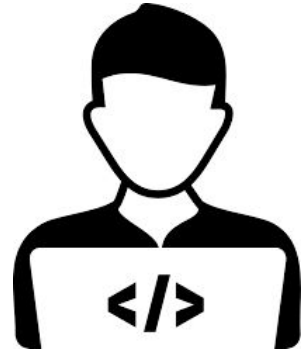
Evaluate its Correctness



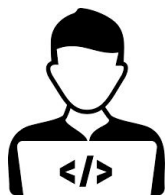
Fix Potential Bugs



Integrate into Codebase



# Research Gap



Prompt



Validate  
Repair



Generate

```
def parse_headers_response_headers(headers):
    """Parse the list of headers and return the list of values (keys, val)
    Split into starting with a
    Parse the list into dictionary
    Finally convert it into
    """
    headers = {}
    for line in headers_string.splitlines():
        key, val = line.split(":")
        headers[key] = val

    return headers

def parse_headers_string(headers_string):
    """Parse the list of headers and return the list of values (keys, val)
    Split into starting with a
    Parse the list into dictionary
    Finally convert it into
    """
    headers = {}
    for line in headers_string.splitlines():
        key, val = line.split(":")
        headers[key] = val

    return headers

headers = {}
for line in headers_string.splitlines():
    key, val = line.split(":")
    headers[key] = val

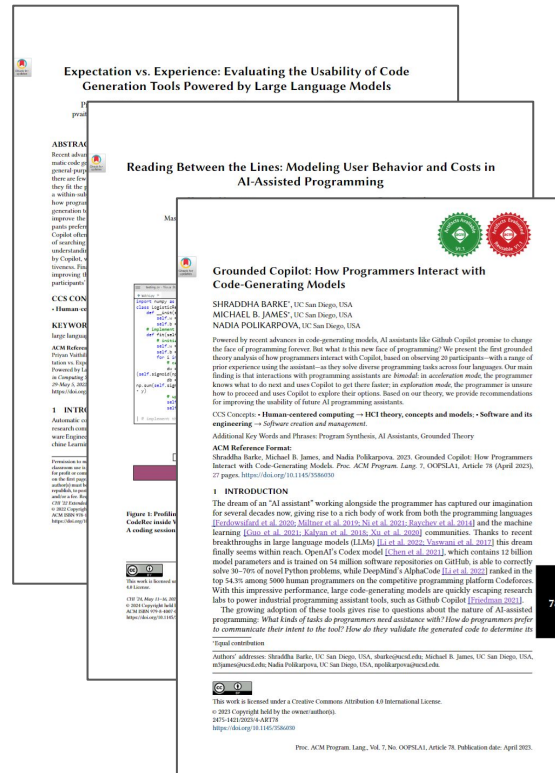
return headers

headers = {}
for line in headers_string.splitlines():
    key, val = line.split(":")
    headers[key] = val

return headers

headers = {}
for line in headers_string.splitlines():
    key, val = line.split(":")
    headers[key] = val

return headers
```



# Research Question

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



Error discovery and repair strategies are crucial for successful human-AI interaction.




Compared to traditional debugging, LLMs can generate different types of errors than humans developers.

# 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?

 Awareness of code provenance impacts developers' behavior when programming, even though they may not always be conscious of such biases.

# 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



# Participants

## University of Notre Dame

28 participants

## Gender

17 male, 11 female

## Education Level

14 graduate, 14 undergraduate

## Majors

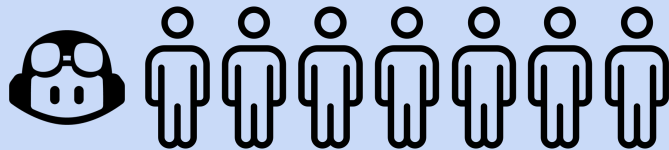
26 in computer science & engineering

## Programming Experience

5.5 years (average)

## GitHub Copilot Usage

8 participants used it before the study

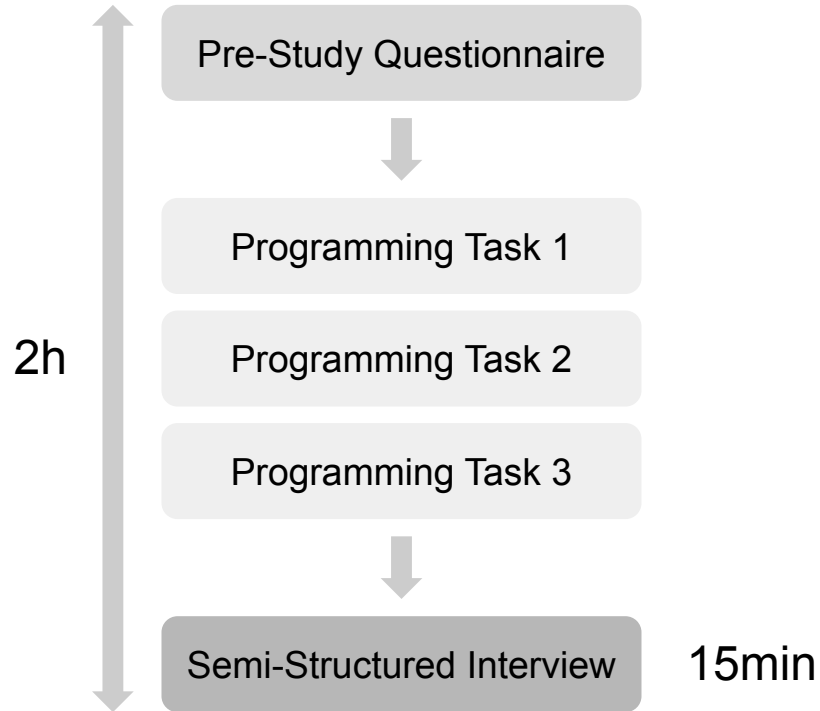


Informed Group (14)

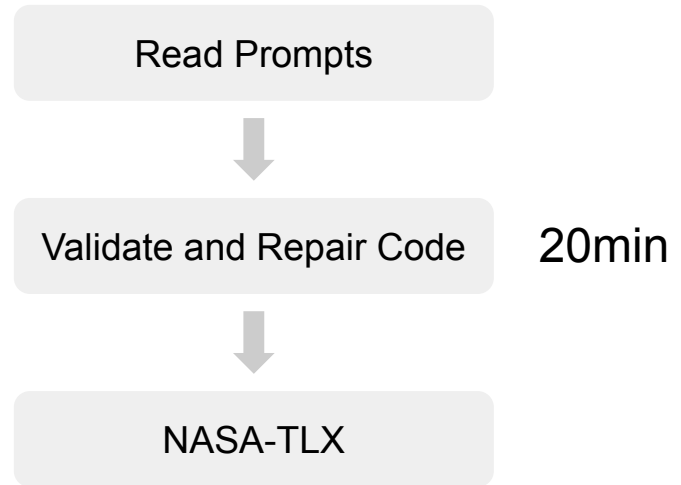


Non-Informed Group (14)

# Study Protocol



For each programming task:



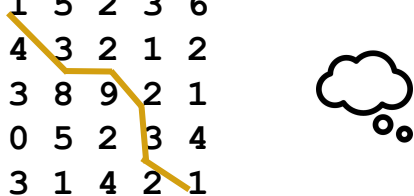
# Programming Tasks

## Task 1. Algorithm Design: Kakamora

*Find the path from top-left to bottom-right such that the sum of numbers is minimized.*

Sample Input

```
5
1 5 2 3 6
4 3 2 1 2
3 8 9 2 1
0 5 2 3 4
3 1 4 2 1
```




Sample Output

```
12
1 3 2 2 3 1
```

Copilot's Solution

```
5
1 5 2 3 6
4 3 2 1 2
3 8 9 2 1
0 5 2 3 4
3 1 4 2 1
```



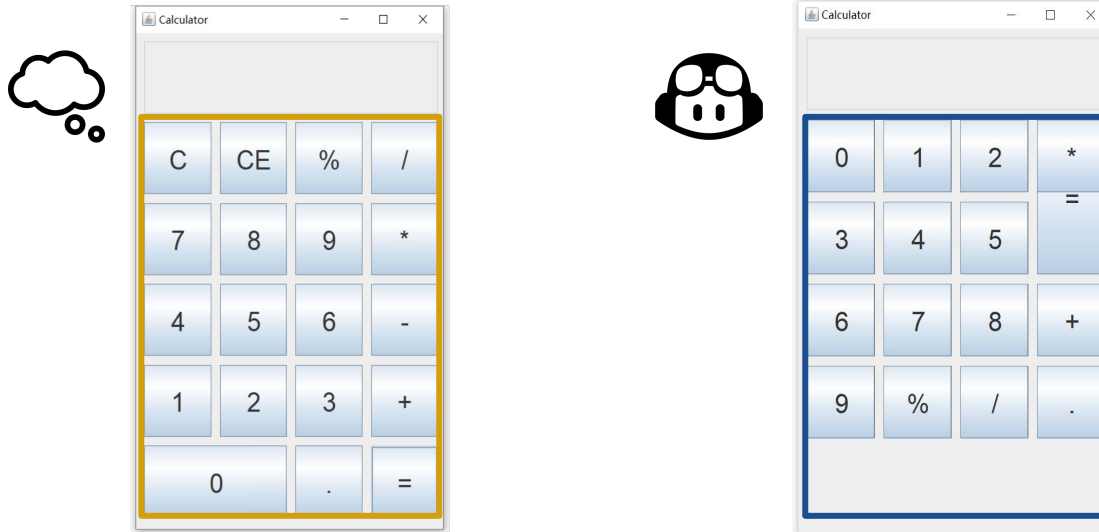
19

```
1 3 3 3 3 2 2 2 2
```

# Programming Tasks

## Task 2. Graphical User Interface (GUI): Calculator

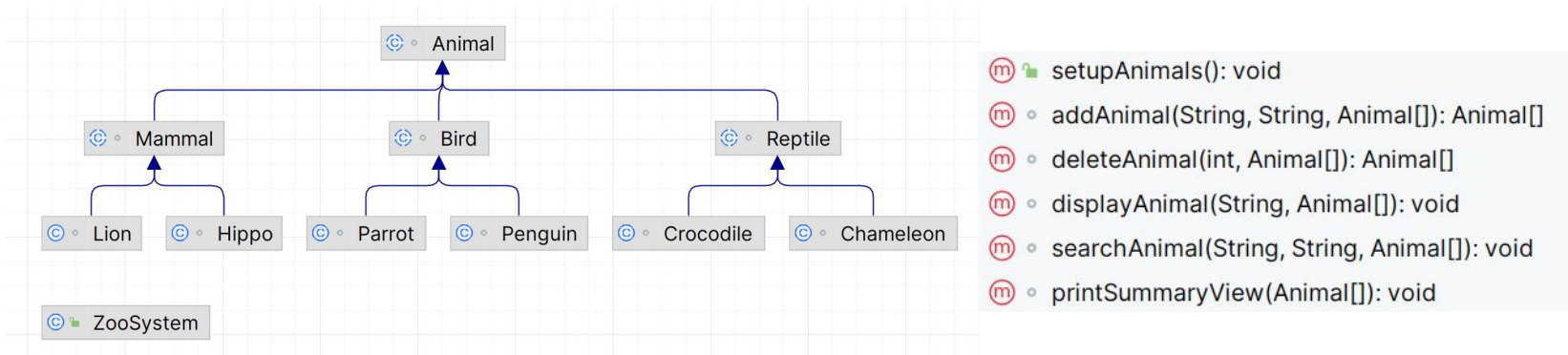
*Use the Java GUI programming API to implement the front end of a calculator.*



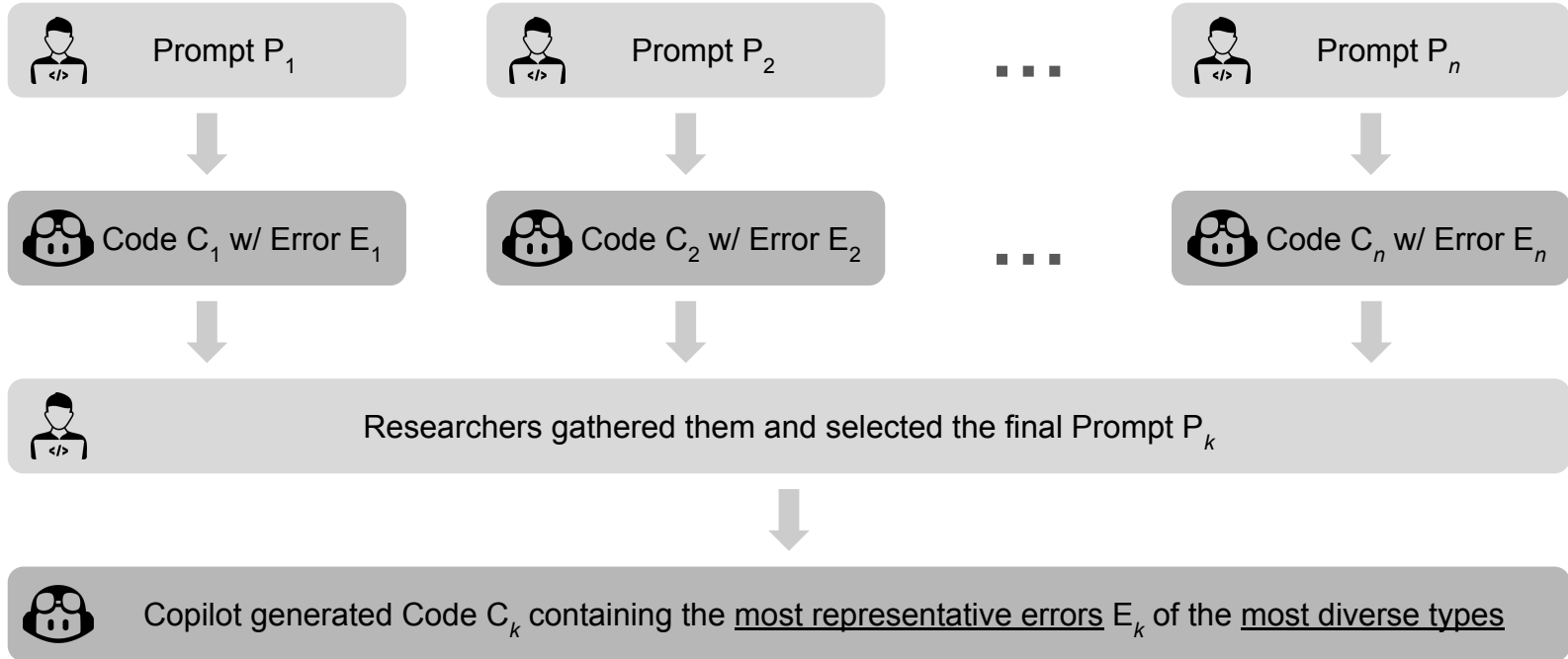
# Programming Tasks

## Task 3. Object-Oriented Programming (OOP): ZooSystem

*Implement a zoo system with inherited animal classes and various management functions.*



# How Code Is Generated: Diverse Error Types



# Bug Taxonomy

TABLE I  
BUG TYPES REPRESENTED IN PROGRAMMING TASKS BASED ON [59].

Task	Subtask	Bug Index	Bug Category
Kakaroma	1.1	231x	Missing Case
	1.2	3226.4	String Manipulation-Insertion
		3126	Illogic Predicates
		231x	Missing Case
Calculator	2.1	6125	Parameter Value
	2.2	614x	Initialization State
ZooSystem	3.1	6125	Parameter Value
	3.2	6112	Wrong Component
		6125	Parameter Value
	3.3	4164	Should be Dynamic Resource
	3.4	6112	Wrong Component
	3.5	413x	Initial, Default Values

```
Lion(String name){  
    super(name);  
    this.maneSize = 0.0;  
    this.colorOfMane = "brown";  
    this.lionType = "African";  
    this.setSpecies("Lion");  
    this.setConservationStatus("Not Extinct");  
    this.setEatingHabits("Carnivores");  
    this.setFeatherColor("All Colors");  
    this.setBeakShape("Short and Conical");  
    this.setEggSize(25);  
}
```

6125 Parameter Value

6112 Wrong Component

# Data Collection

## 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="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="1696217111">
</action>
<action id="RenameElement" path="/src/ABC.java" timestamp="1696217111">
</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>
```

## Eye Tracking

```
16666666666666666666 gaze_point_y="0.17407407407407408" gaze_validity="1.0"
3662841796875" pupil_validity="1.0"/>
54166666666666666666 gaze_point_y="0.17407407407407408" 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

```
import java.util.Map;
public class TwoSum {
    public static int[] findTwoSum(int[] nums, int target) {
        Map<Integer, Integer> numMap = new HashMap<>(); //use hash
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (numMap.containsKey(complement)) {
                return new int[]{numMap.get(complement), i};
            }
            numMap.put(nums[i], i);
        }
        return null;
    }
    public static void main(String[] args) {
        int[] nums = {2, 7, 11, 15};
        int target = 9;
        int[] result = findTwoSum(nums, target);
        if (result != null) {
            System.out.println("Indices found: " + result[0] + ", " + result[1]);
        } else {
            System.out.println("No solution found");
        }
    }
}
```

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

# Paper

## CodeGRITS: A Research Toolkit for Tracking

Ningzhi Tang<sup>†</sup>, Junwen An<sup>†</sup>, Yu Huang<sup>†</sup>, Collin McElrath<sup>†</sup>, and Benoit B. Bessière<sup>†</sup>, et al.  
<sup>†</sup>University of Notre Dame  
<sup>†</sup>Vanderbilt University

### ABSTRACT

Traditional methodologies for exploring programmers' behaviors have primarily focused on capturing their actions within the Integrated Development Environment (IDE), offering limited view into their cognitive processes. Recent emergent work started using eye-tracking techniques in software engineering (SE) research. However, the lack of tools specifically designed for coordinated data collection poses technical barriers and requires significant effort from researchers who wish to combine these two complementary approaches. To address this gap, we present CodeGRITS, a plugin specifically designed for SE researchers. CodeGRITS is built on top of IntelliJ's SDK, with wide compatibility with the entire family of JetBrains IDEs to track developers' IDE interactions and eye gaze data. CodeGRITS also features various practical features for SE research (e.g., activity labeling) and a real-time API that provides interoperability for integration with other research instruments and developer tools. The demo video is available at [https://youtu.be/d\\_YuJWZNM](https://youtu.be/d_YuJWZNM).

### KEYWORDS

IDE Extension/Plugin, Developer Behavior Analysis, Eye Tracking

### 1 INTRODUCTION

Tracking developers' programming behavior provides valuable insights into how they engage in the software development process [19, 23] and helps evaluate and improve the usability of programming language features and tools in software engineering (SE) research [11, 26]. Traditional approaches focus mainly on tracking developers' interactions with the integrated development environment (IDE), such as keystrokes, code changes, and IDE-specific commands [11, 23]. However, while these approaches can identify "what a programmer did," they are limited in explaining "why they did it." Previous research relies mainly on surveys and interviews to understand what developers were thinking and why they made

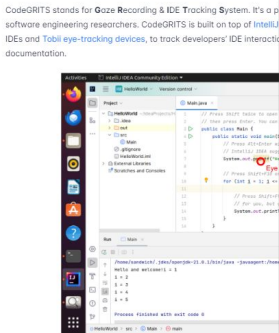
<sup>†</sup>Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from [permissions@acm.org](mailto:permissions@acm.org).

©2024 Copyright 2024, April 26–30, 2024, Lisbon, Portugal.  
 ©2024 Copyright held by the owner(s). Publication rights licensed to ACM.  
 ACM ISBN 978-1-4503-9318-4/24/04...\$15.00.  
<https://doi.org/10.1145/3670793.3680077>

## Welcome to CodeGRITS

NEWS! We would present CodeGRITS at ICSE 2024 Demo Track. Welcome!



The data collected by CodeGRITS can be used by empirical SE research eye gaze. CodeGRITS also provides a **real-time data API** for future plug-in support tools.

CodeGRITS is still in its developmental stage as a research tool particularly for those involved in empirical software engineering, through [GitHub Issue](#) for any suggestions or issues, siding in it.

For any inquiries, please email us at [ntang@nd.edu](mailto:ntang@nd.edu) or [jan2@nd.edu](mailto:jan2@nd.edu) and hesitate to email us for setup support. We are delighted to provide environment.

### Cross-platform and Multilingual Support

- CodeGRITS provides cross-platform support for Windows, macOS, JetBrains IDEs, including IntelliJ IDEA, PyCharm, WebStorm, etc.
- CodeGRITS could extract the abstract syntax tree (AST) structure supports them, including Java, Python, C/C++, JavaScript, etc.

macOS Support

# Website

# Source Code

CodeGRITS stands for Gaze Recording & IDE Tracking System. It's a plugin developed by the [Software Engineering Research 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 [Toolt eye-tracking devices](#) to track developers' IDE interactions and eye gaze data.

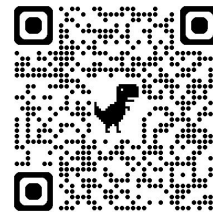
File Name	Language	Last Update
github/workflows	update:retype.yml	6 months ago
idea	doc-mac	2 weeks ago
run	doc-mac	6 months ago
gradle/wrapper	first commit	8 months ago
site	doc-mac	2 weeks ago
src/main	doc-mac	2 weeks ago
github	first commit	8 months ago
AUTHORS	type	4 months ago
LICENSE	website & readme	4 months ago
README.md	macos	last month
build.gradle.kts	setting	4 months ago
gradle.properties	first commit	8 months ago
gradlew	first commit	8 months ago
gradle.bat	first commit	8 months ago
retype.yml	javaDoc	3 months ago
settings.gradle.kts	rename & start/stop/pause/resume track	4 months ago

### CodeGRITS

Windows | macOS | Linux | Android | iOS | Desktop | Web | Activated | SWT | JavaFX | SWT

CodeGRITS stands for Gaze Recording & IDE Tracking System. It's a plugin developed by the [Software Engineering Research 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 [Toolt eye-tracking devices](#) to track developers' IDE interactions and eye gaze data.

The [website's](#) source code is stored in the `/website` folder of this repository and deployed via GitHub Pages. The [JavaDoc](#) documentation is located in the `/javadoc` folder of this repository. They are archived together with the



[codegrits.github.io/](https://codegrits.github.io/)  
**CodeGRITS**

# High-Level Behavior Aggregation

TABLE II  
CATEGORIZATION OF DEVELOPER BEHAVIOR EMERGED FROM IDE AND EYE TRACKING DATA.

Index	Behavior	Tracking Data
1	Reading Document	Consecutive fixations on the instructional document
2	Reading Code	Consecutive fixations on the code
3	Reading Comment	Consecutive fixations on the comment
4	Switching Files	Opening, closing, or changing the selection of a file
5	Scrolling	Scrolling a file via mouse wheel, arrow keys, or touchpad gestures
6	Tracing Code	Searching tokens, finding usages or going to declarations
7	Running for Output	Running the class to obtain execution output
8	Employing Debugger	Utilizing debugger and its corresponding features (e.g., toggling breakpoints)
9	Invoking Copilot	Accepting, rejecting, or browsing code generated from Copilot
10	Utilizing Clipboard	Copying, cutting, or pasting contents
11	Keystrokes Typing	Typing characters using keystrokes

## RQ1. Perceptions and Strategies of Developers

**LLM-generated code performs well in terms of coding style and readability.**

*“[LLM-generated code] does follow **human formatting guidelines**.” (P21)*

*“It writes **better** variable names or method names.” (P25)*

*“It contains **more detailed comments** compared to human-written code.” (P14)*

## RQ1. Perceptions and Strategies of Developers


LLM-generated code performs well in terms of coding style and readability.  
**However, it makes mistakes that are uncommon for human developers.**

*"I was really **confused** when I saw the **hardcoded '1', '2', and '3'.**" (P21)*

```
Sample Input
5
1 5 2 3 6
4 3 2 1 2
3 8 9 2 1
0 5 2 3 4
3 1 4 2 1

Sample Output
12
1 3 2 2 3 1
```

```
String path = "";
while (i != 0 || j != 0) {
    if (i == 0) {
        path = "2 " + path;
        j--;
    } else if (j == 0) {
        path = "3 " + path;
        i--;
    } else if (table[i - 1][j] < table[i][j - 1]) {
        path = "3 " + path;
        i--;
    } else {
        path = "2 " + path;
        j--;
    }
}
path = "1 " + path;
```

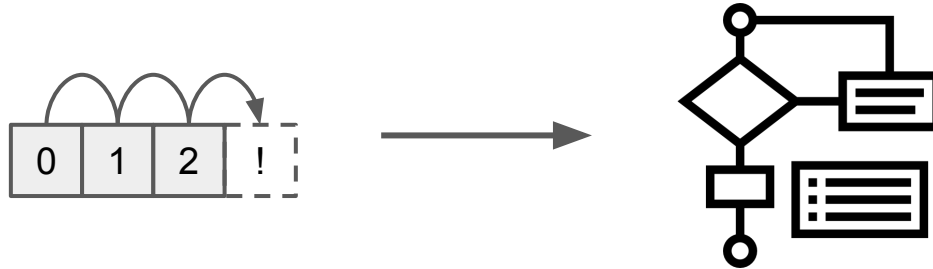


## RQ1. Perceptions and Strategies of Developers

**Developers have shifted their attention focus from details to structure.**

*“Humans are more error-prone than Copilot when it comes to **details** [...]”*

*“[...] I don’t think Copilot can generate really complicated **logic structure**. LLM-generated similar code tends to be either all correct or all incorrect.” (P25)*

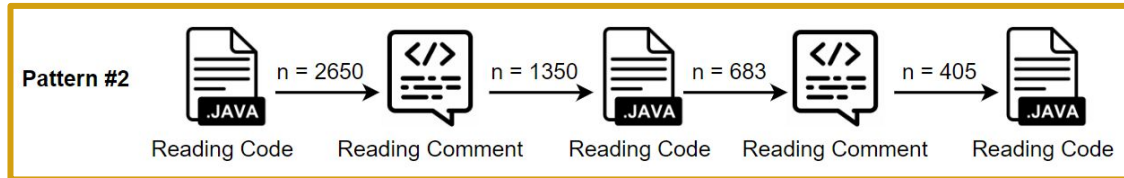
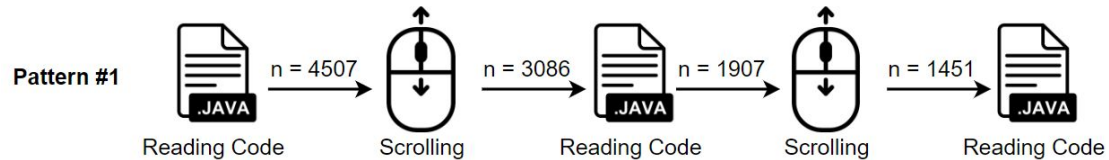


## RQ1. Perceptions and Strategies of Developers

Developers **switch their fixations** consecutively between code and prompts.

*“[...] **disambiguate the mismatch** between their expectation and code output.” (P14)*

*“Switching between instructions and code is **annoying and challenging**.” (P3)*



## RQ1. Perceptions and Strategies of Developers

Developers use Copilot to generate comments to facilitate understanding.

*“If a line of code has a bug, generating comments from it will help me **figure it out**.” (P25)*

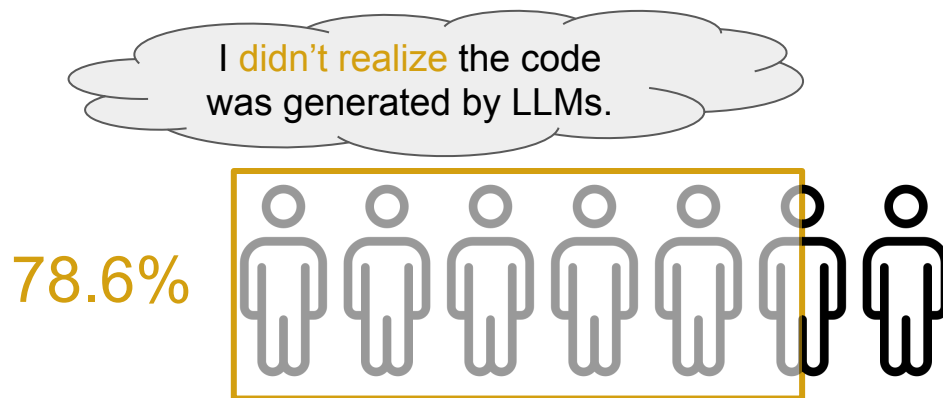
```
public class BinarySearch {  
    // Returns the index of the element if found, otherwise returns -1  
    no usages new *  
    public static int binarySearch(int[] arr, int element) {  
        int left = 0, right = arr.length - 1;  
        while (left <= right) {  
            int mid = left + (right - left) / 2;  
            if (arr[mid] == element) return mid;  
            if (arr[mid] < element) left = mid + 1;  
            else right = mid - 1;  
        }  
        return -1;  
    }  
}
```

**Inline Comments**



## RQ2. Effects of Code Provenance Knowledge

If uninformed, developers may not distinguish code provenance.

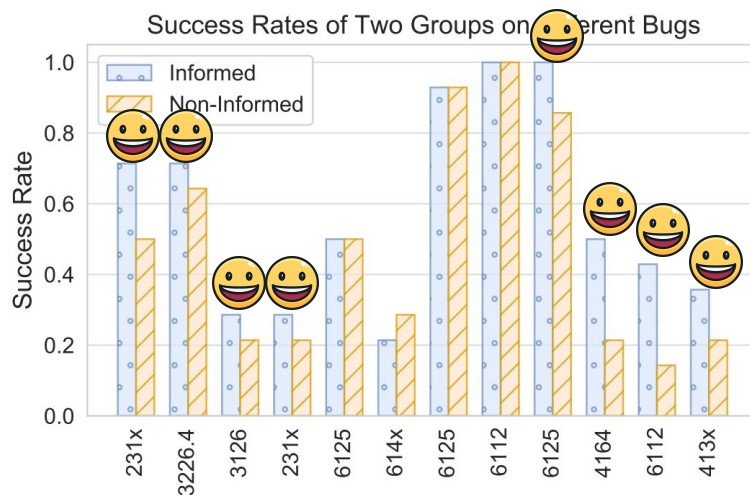


## RQ2. Effects of Code Provenance Knowledge

If uninformed, developers may not distinguish code provenance.

If informed, developers performs better at code validation and repair.

Bug fix rate:  $0.577 > 0.446$ . Performed better on 8/13 bugs.



## RQ2. Effects of Code Provenance Knowledge

If informed, developers use Copilot more and Clipboard less.



Copilot

Substitute  
→



Cut



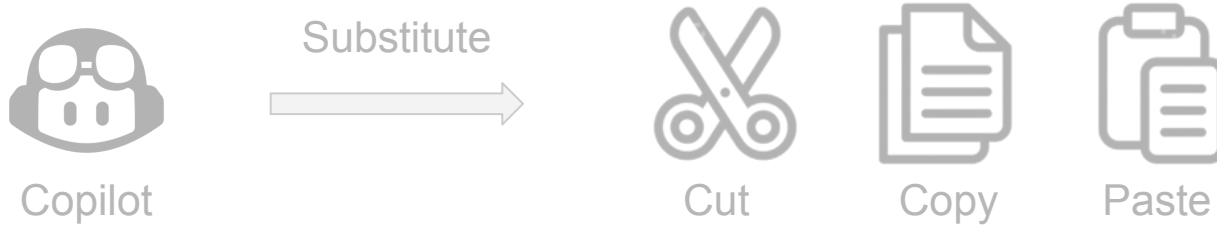
Copy



Paste

## RQ2. Effects of Code Provenance Knowledge

If informed, developers use Copilot more and Clipboard less.



If informed, developers trace code more frequently.



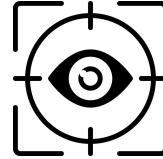
## RQ2. Effects of Code Provenance Knowledge

If informed, developers experienced higher cognitive workload.



**6.39 > 5.60**

Self-Reported Effort  
(via NASA-TLX)



**254.3 > 206.7**

**0.493 > 0.438**

Fixation Time &  
Average Fixation Duration

# Implications

LLMs make distinct types of mistakes and developers have shifted attention from details to structure.



Adaptive systems specifically designed based the unique characteristics of LLM-generated code.

Usage of Copilot to generate inline comments, but with frequent switching between code and prompts.



Improved interfaces to support leveraging LLM to understand and modify the code, while reducing the switching costs.

Developers may not realize the code provenance, yet it impacts their behavior and performance.



Detecting and informing developers of the code provenance information would enhance interactions with LLMs.

# Future Work

## Enhancing Interactions with LLM

### Code Understanding & Modification

~~Textual Representation~~



Tree Representation

Multi-Level

Syntax-Aware

```
// 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**

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**

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





**E**

# Developer Behaviors in Validating and Repairing LLM-Generated Code Using IDE and Eye Tracking

Ningzhi Tang\*, Meng Chen\*, Zheng Ning, Aakash Bansal, Yu Huang, Collin McMillan, Toby Jia-Jun Li  
 {ntang, mchen24, zning, abansal1, cmc, toby.j.li}@nd.edu, yu.huang@vanderbilt.edu



**Ningzhi Tang**

 nztang.com  
 ntang@nd.edu  
 @TangNingzhi  
 ningzhi\_tang

LLMs make distinct types of mistakes and developers have shifted attention from details to structure.



Adaptive systems specifically designed based the unique characteristics of LLM-generated code.

Usage of Copilot to generate inline comments, but with frequent switching between code and prompts.



Usage of Copilot to generate inline comments, but with frequent switching between code and prompts.

Developers may not realize the code provenance, yet it impacts their behavior and performance.



Detecting and informing developers of the code provenance information would enhance interactions with LLMs.



codegrits.github.io/  
CodeGRITS



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.

