



Refactoring with LLMs: Lessons Learned

Danny Dig, Abhiram Bellur


University of Colorado, JetBrains Research



Assistant augments our capacity

11:49 📶 🔋

[← Workouts](#) Sat, Sep 9 📤

 Outdoor Cycle
Open Goal
8:42 AM–9:34 AM
📍 Cottonwood Heights



Workout Details [Show More](#)

Workout Time
0:51:15

Distance
9.53MI

Active Calories
1,148CAL

Total Calories
1,256CAL

Elevation Gain
2,599FT

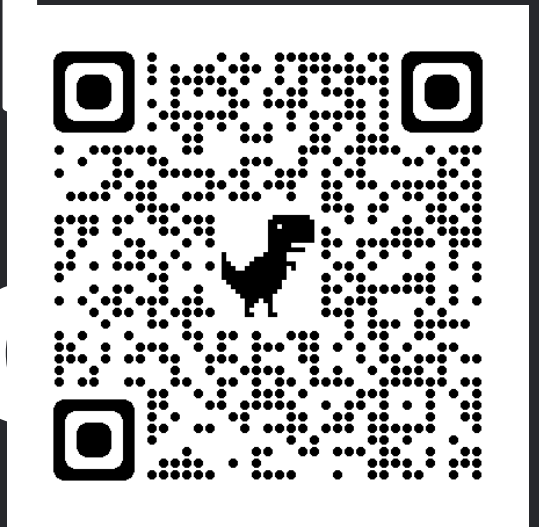
Avg. Speed
11.1MPH

Avg. Heart Rate
142BPM

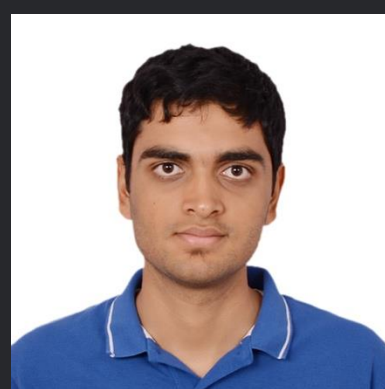


85 Nm, 500 Wh

Next Generation Refactoring: LLM Insights and IDE for ExtractMethod



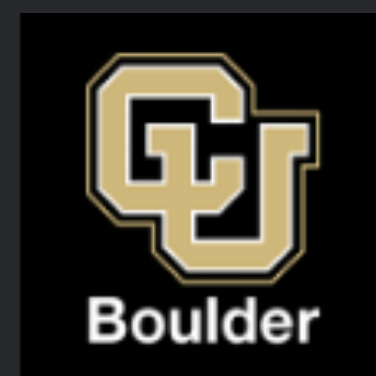
Dorin
Pomian



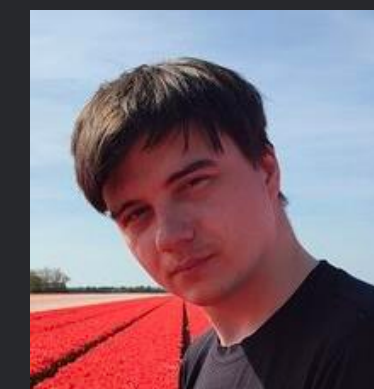
Abhiram
Bellur



Malinda
Dilhara



Zarina
Kurbatova



Andrey
Sokolov



Egor
Bogomolov



Timofey
Bryksin



Danny
Dig



Long Methods In Codebases

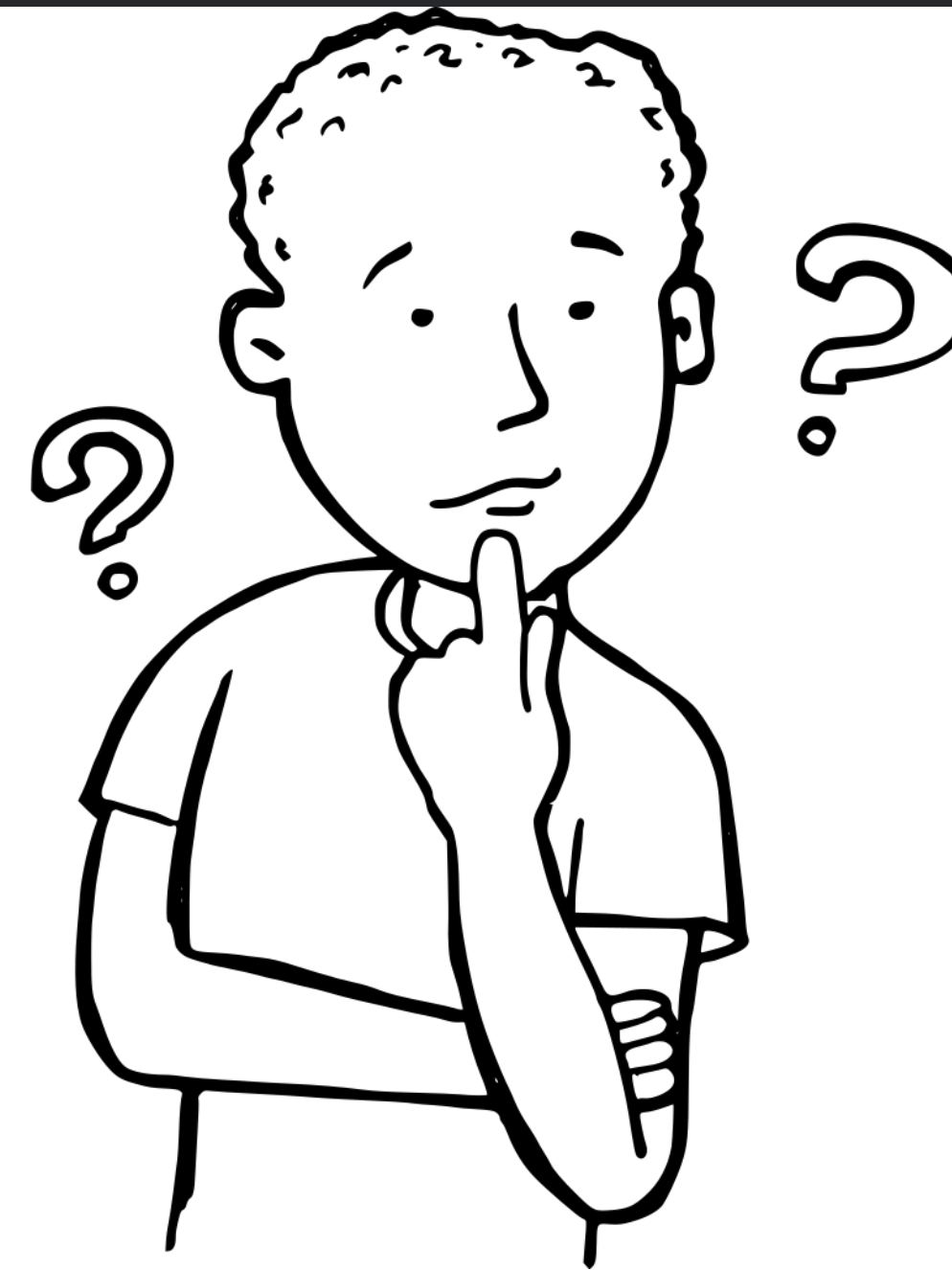
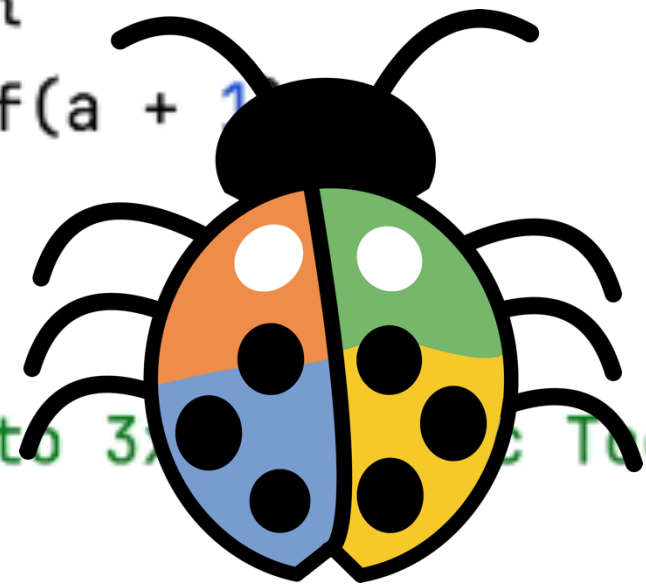
Abhiram98

```
public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);
    board = new String[9];
    turn = "X";
    String winner = null;

    for (int a = 0; a < 9; a++) {
        board[a] = String.valueOf(a + 1);
    }

    System.out.println("Welcome to 3x3 Tic Tac Toe.");
    printBoard();

    System.out.println(
        "X will play first. Enter a slot number to place X in:"
```



Extract Method Refactoring

```
74     out.writeUTF(myInterface);
75 }
76 // Write myFields
77 int fieldsCount = 0;
78 for (JvmField field : jvmClass.getFields()) fieldsCount++;
79 DataInputOutputUtil.writeINT(out, fieldsCount);
80 for (JvmField field : jvmClass.getFields()) {
81     writeJvmField(field, out);
82 }
83
84 // Write myMethods
85 int methodCount = 0;
86 for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
87 DataInputOutputUtil.writeINT(out, methodCount);
88 for (JvmMethod jvmMethod : jvmClass.getMethods()) {
89     writeJvmMethod(jvmMethod, out);
90 }
91
92 // Write AnnotationTargets
93 int elemTypeCount = 0;
94 for (ElemType elemType : jvmClass.getAnnotationTargets()) elemTypeCount++;
95 DataInputOutputUtil.writeINT(out, elemTypeCount);
96 for (ElemType elemType : jvmClass.getAnnotationTargets()) {
97     writeElemType(elemType, out);
98 }
99
100 if (jvmClass.getRetentionPolicy() != null) {
```

1. Original Method

```
103 @ private static void writeMethods(JvmClass jvmClass, DataOutput out) throws IOException {
104     int methodCount = 0;
105     for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
106     DataInputOutputUtil.writeINT(out, methodCount);
107     for (JvmMethod jvmMethod : jvmClass.getMethods()) {
108         writeJvmMethod(jvmMethod, out);
109     }
110 }
```

2. Extracted Method

```
80     for (JvmField field : jvmClass.getFields()) {
81         writeJvmField(field, out);
82     }
83
84     // Write myMethods
85     writeMethods(jvmClass, out);
86
87     // Write AnnotationTargets
88     int elemTypeCount = 0;
89     for (ElemType elemType : jvmClass.getAnnotationTargets()) elemT
```

3. Call Site

Current Extract Method Workflow in IntelliJ



JetBrains' IntelliJ IDEA has extract method capabilities



Semi-automated process



No automatic recommendations

```
63 static void writeJvmClass(JvmClass jvmClass, DataOutput out) throws IOException {
64     writeJVMClassNode(jvmClass, out);
65     out.writeUTF(jvmClass.getSuperFqName());
66     out.writeUTF(jvmClass.getOuterFqName());
67     // Write myInterfaces;
68     int interfacesCount = 0;
69     for (String myInterface : jvmClass.getInterfaces()) {
70         interfacesCount++;
71     }
72     DataInputOutputUtil.writeINT(out, interfacesCount);
73     for (String myInterface : jvmClass.getInterfaces()) {
74         out.writeUTF(myInterface);
75     }
76     // Write myFields
77     int fieldsCount = 0;
78     for (JvmField field : jvmClass.getFields()) fieldsCount++;
79     DataInputOutputUtil.writeINT(out, fieldsCount);
80     for (JvmField field : jvmClass.getFields()) {
81         writeJvmField(field, out);
82     }
83
84     // Write myMethods
85     int methodCount = 0;
86     for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
87     DataInputOutputUtil.writeINT(out, methodCount);
88     for (JvmMethod jvmMethod : jvmClass.getMethods()) {
89         writeJvmMethod(jvmMethod, out);
90     }
91
92     // Write AnnotationTargets
93     int elemTypeCount = 0;
94     for (ElemType elemType : jvmClass.getAnnotationTargets()) elemTypeCount++;
95     DataInputOutputUtil.writeINT(out, elemTypeCount);
}
```

Extract Method Research



Many research tools for recommending fragments to extract

- *JDeodorant*

- *JExtract*

- *LiveREF*

- *REMS*

- *GEMS*

- *SEMI*



Optimize software quality metrics



Generate refactorings that do not align with developers' preferences

Large Language Models (LLMs) + Refactoring

Corpus of 2,849 real-life methods:



LLMs are creative and prolific: 12,387 Extract Method suggestions (averaging 4 suggestions per method)



45.7% of the suggestions may be invalid, potentially resulting in non-compiling code



16.6% of suggestions are not useful (e.g. one liners, or entire method body)



Key Idea: LLMs for recommendation + IDE for safe execution

Our solution: *EM-Assist*

IntelliJ IDEA plugin implementation

Leverage creative capabilities of LLMs

Use static analysis techniques to filter, further enhance, and rank LLM-provided suggestions

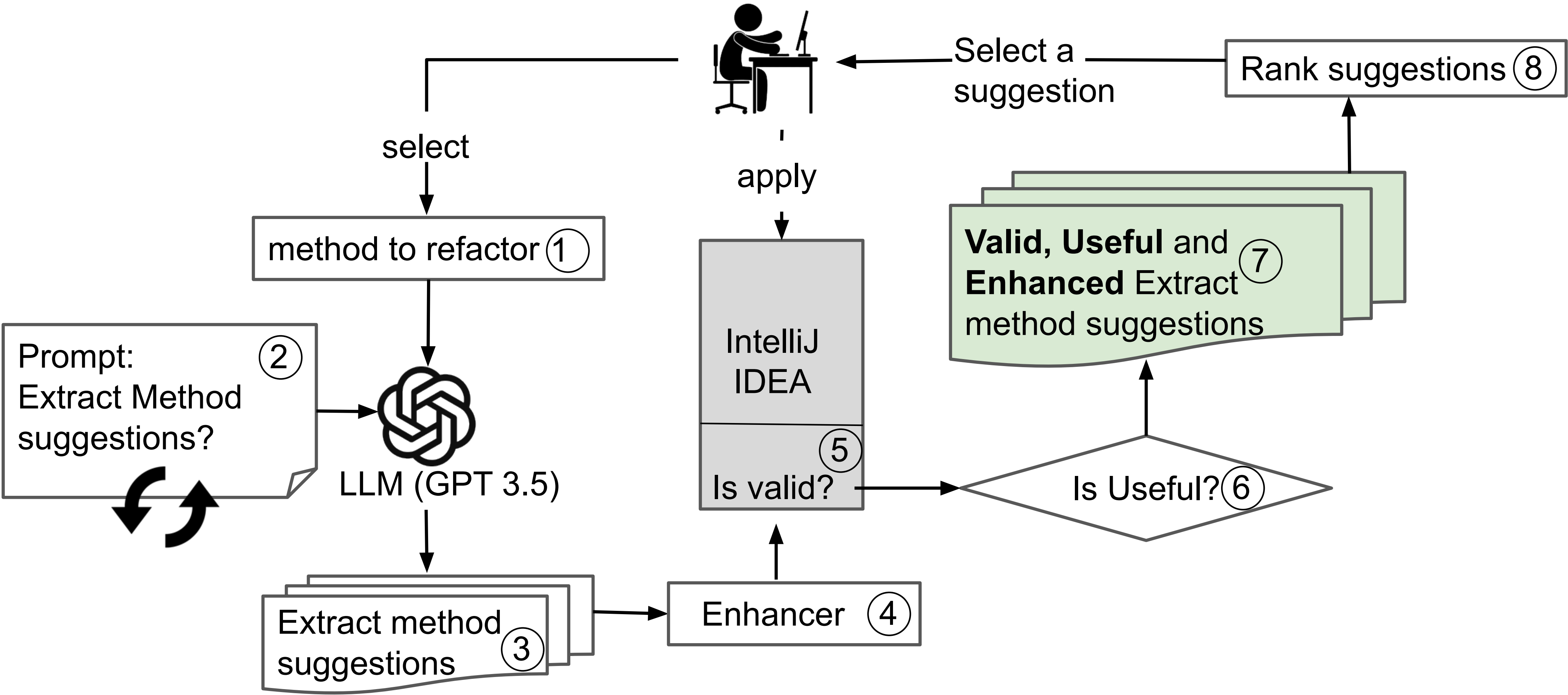
Utilize the full power of a state-of-the-practice commercial IDE, IntelliJ IDEA, to apply refactorings safely



Demo



EM-Assist Workflow



EM-Assist Evaluation Results

Oracle of actual 1,752 extract method refactorings from OSS

- EM-Assist achieved **53%** recall rate
- Compared to
 - 39% recall rate by JExtract (best in class using static analysis)
 - 5% recall rate by LiveREF

18 developers participated in usability survey, **94%** gave a positive rating:

"Thank you for interesting suggestions! Hope to see this in production."

"These suggestions made me look at this code with new eyes, and I will refactor it."

LLM-Powered Move Method Assistant

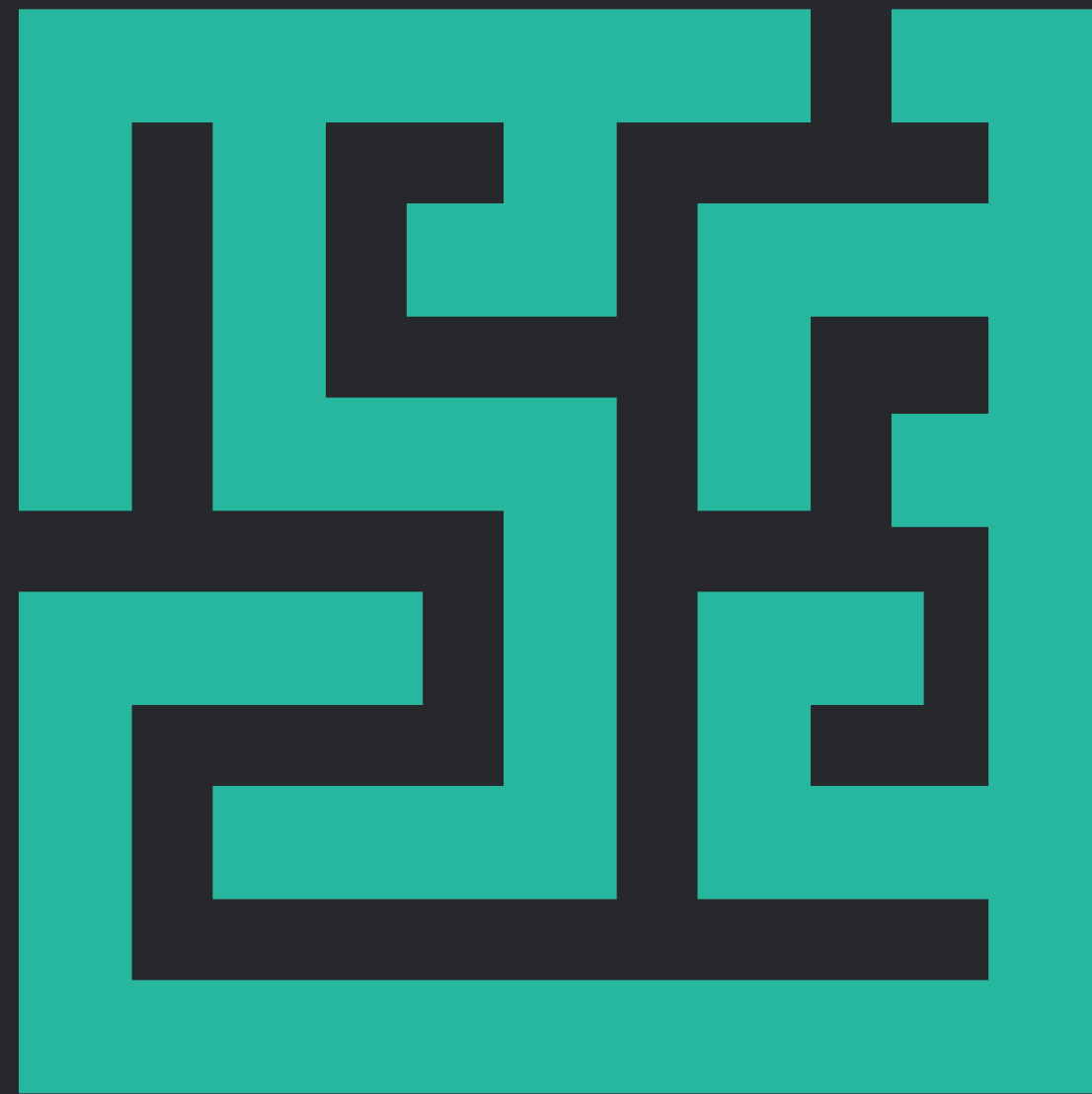
Move Method Refactoring



Solution to feature envy!

<https://refactoring.guru/smells/feature-envy>

Challenges for Move Method



Challenges:

- determine which method is out of place
- find a suitable Target class

Global project understanding

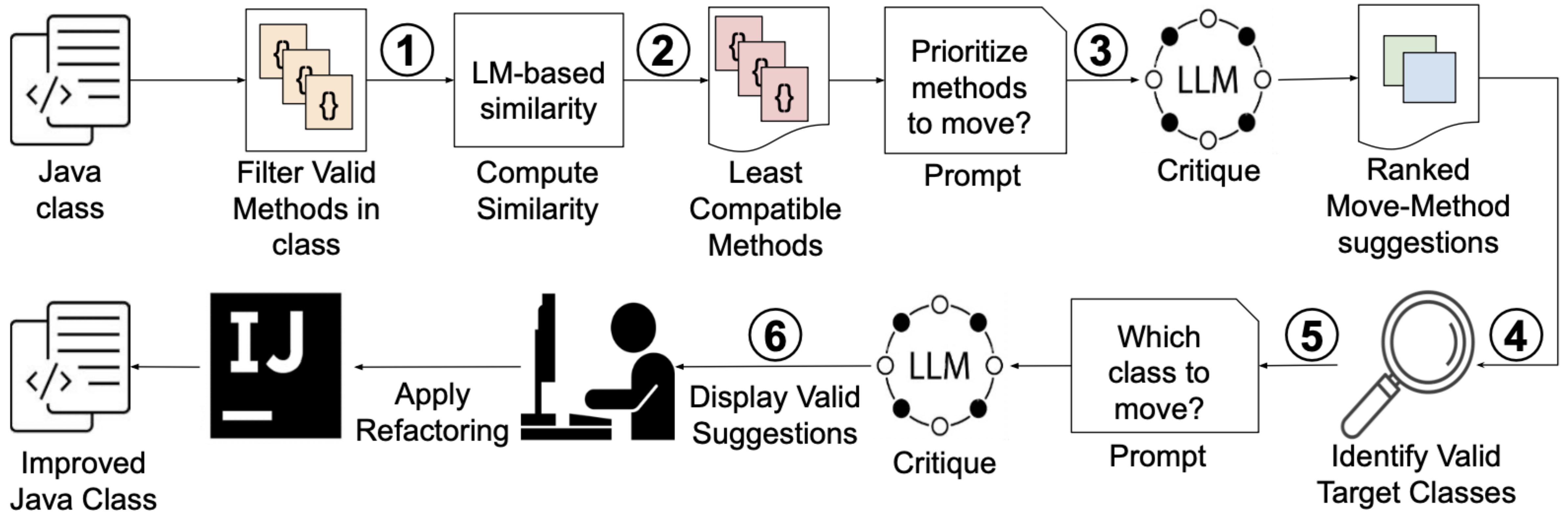


LLM + Vector embeddings + IDE

Demo

```
74 // To validate the job configuration, see:
75 * List<ConfigValue> configValues = defs.validate(props);
76 * // The {@link ConfigValue} contains updated configuration information given the current configuration values.
77 * </pre>
78 * <p/>
79 * This class can be used standalone or in combination with {@link AbstractConfig} which provides some additional
80 * functionality for accessing configs.
81 public class ConfigDef { 1 inheritor 1 Liquean Pei +49
82
83     private static final Pattern COMMA_WITH_WHITESPACE = Pattern.compile(regex: "\\s*,\\s*"); 1 usage
84
85     /**
86      * A unique Java object which represents the lack of a default value.
87      */
88     public static final Object NO_DEFAULT_VALUE = new Object();
89
90     private final Map<String, ConfigKey> configKeys; 19 usages
91     private final List<String> groups; 8 usages
92     private Set<String> configsWithNoParent; 5 usages
93
94     public ConfigDef() { 1 Shikhar Bhushan +1
95         configKeys = new LinkedHashMap<>();
96         groups = new LinkedList<>();
97         configsWithNoParent = null;
98     }
99
100     @ public ConfigDef(ConfigDef base) { 1 Ewen Cheslack-Postava +2
101         configKeys = new LinkedHashMap<>(base.configKeys);
102         groups = new LinkedList<>(base.groups);
103         // It is not safe to copy this from the parent because we may subsequently add to the set of configs and
104         // invalidate this
105         configsWithNoParent = null;
106     }
107
108     /**
109      * Returns unmodifiable set of properties names defined in this {@linkplain ConfigDef}
110      *
111      * @return new unmodifiable {@link Set} instance containing the keys
112      */
113     public Set<String> names() { return Collections.unmodifiableSet(configKeys.keySet()); }
```


Workflow



Results



Corpus of 208 refactorings performed by OSS developers

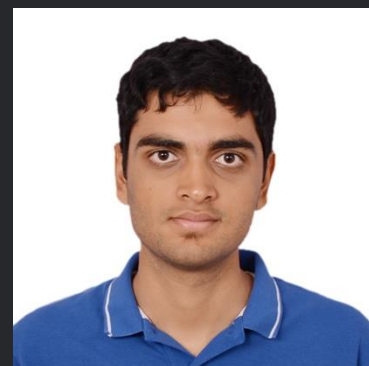
- Recall **82%**
- **4x** better than previous best-in-class tools

FSE'24 Research Track

Unprecedented Code Change Automation: The Fusion of LLM and Transformation by Example



Malinda Dilhara



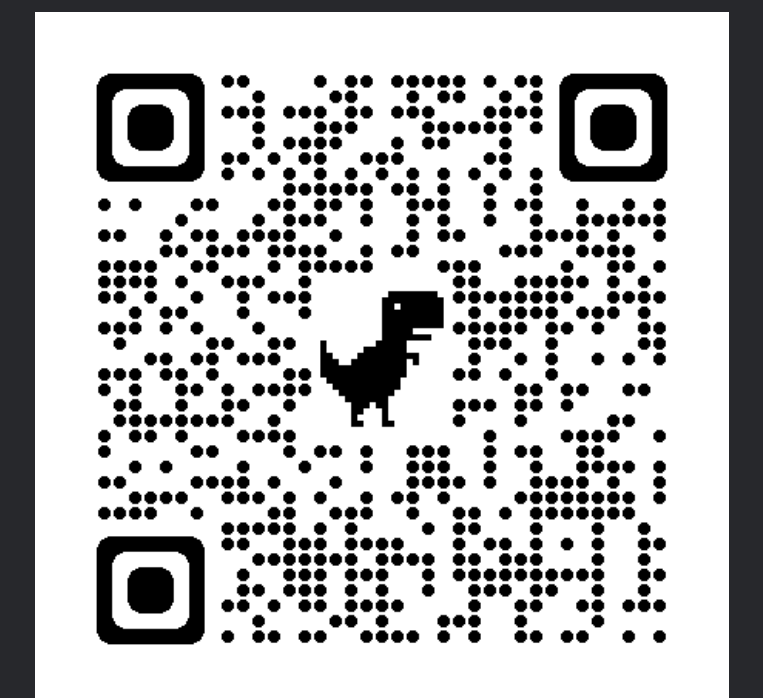
Abhiram Bellur



Timofey Bryksin

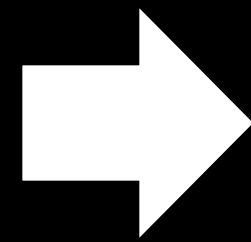


Danny Dig



Code change pattern (CPAT)

```
number = 0  
for x in intArray:  
    number = number + x
```




```
number = numpy.sum(intArray)
```

Commit c8b28432 in GitHub project NifTK/NiftyNet



Transformation By Example

 Learn coding best practices from open-source repos and transplant these into other code

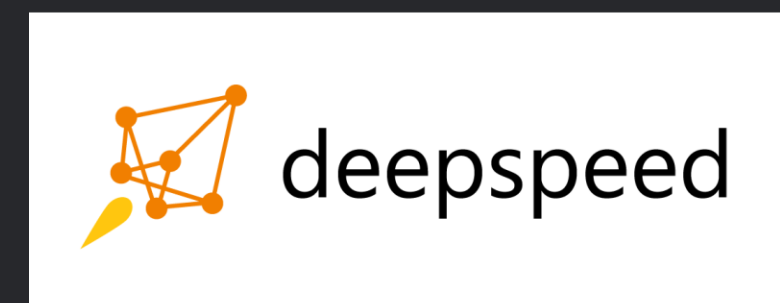
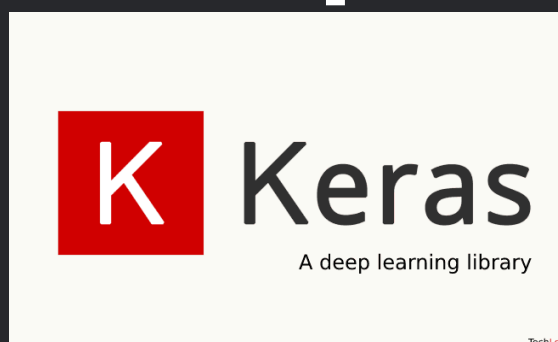
 Cannot apply these to new sites unless the code is exactly the same



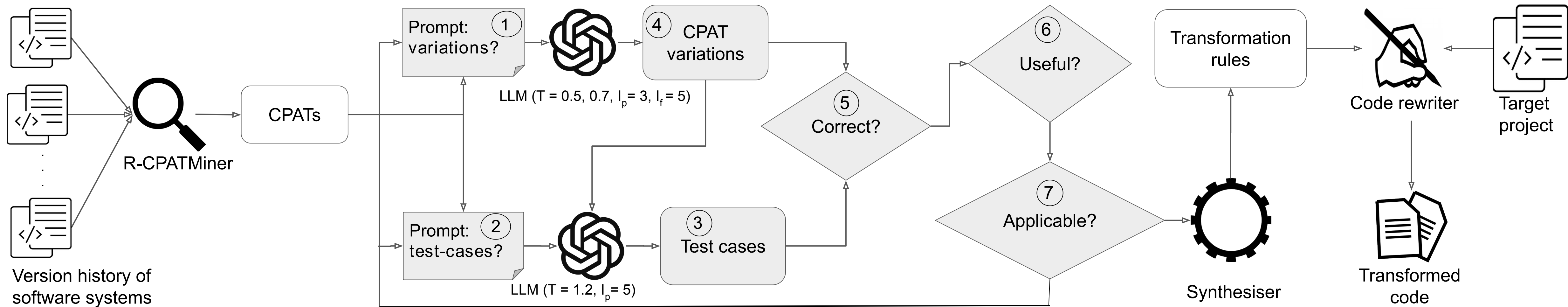
Use LLM to generate many code variants, we validate automatically and apply suggestions to new locations

14x improvements over previous state of the art approach

We contributed to famous open-source projects, they accepted 83% of our suggestions



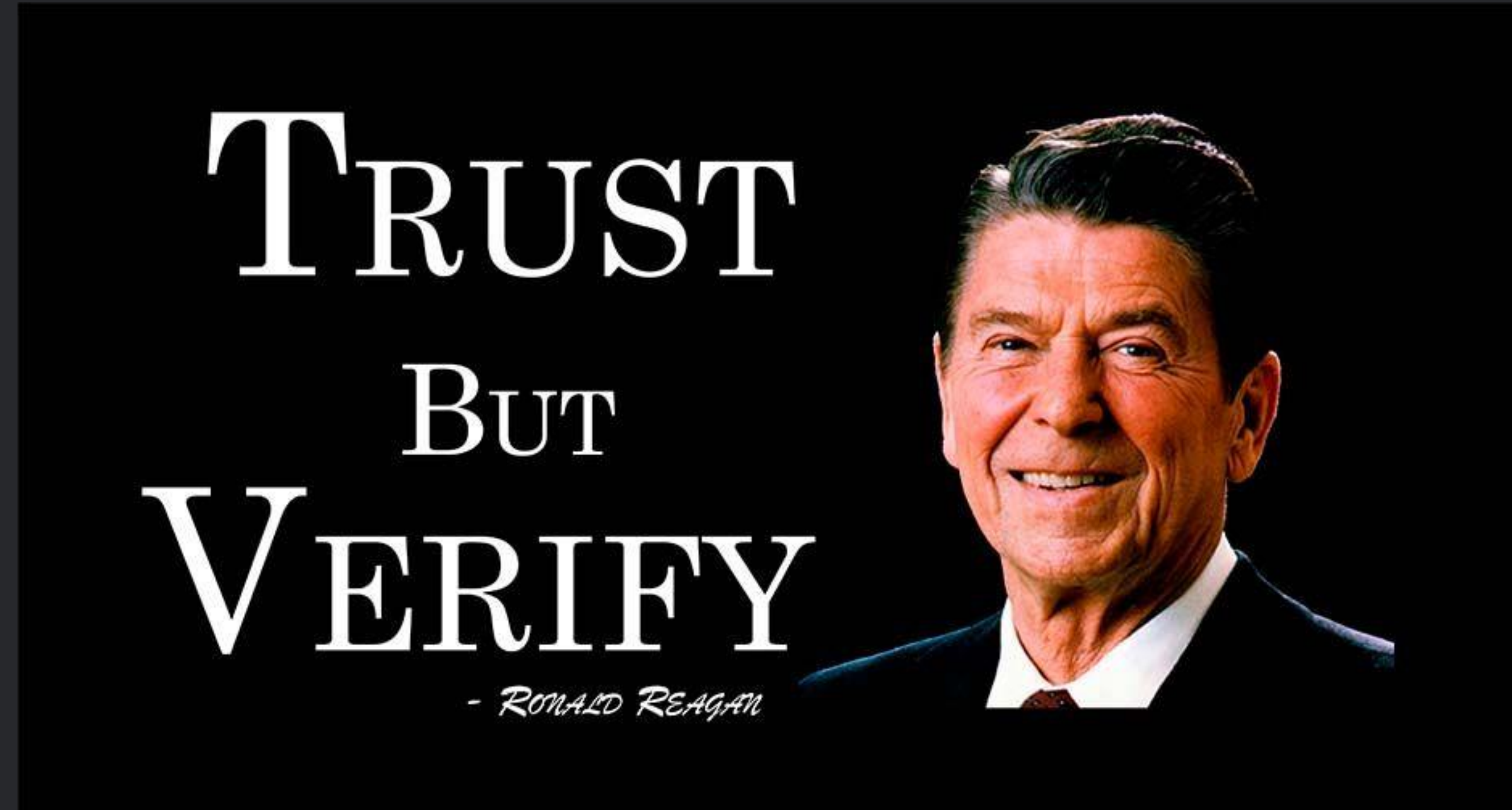
Under the hood: PyCraft



Lessons Learned

LLMs are Prolific but with High rate of hallucinations:

- ExtractMethod: 73% rate of hallucinations
- MoveMethod 80% hallucinations
- PyCraft: 65% hallucinations
- Unit tests: 35% hallucinations



Do what LLM suggests, not what they do => need for powerful validators

- remove hallucinations automatically reusing static analysis from the IDE (e.g., refactoring precondition)
 - Where else can we reuse the IDE as validator?
- new static analysis
- dynamic analysis: generated small unit tests in PyCraft, used original code variant as validator

Lessons Learned

Precise prompt for higher quality suggestions

- append line numbers for the code input
- ask LLM to give you precise response using line numbers
- ask LLM to specify the output in structured format (JSON): useful if the output is consumed by other tools

Few-shot learning worked best for both EM-Assist and PyCraft

For MoveMethod-Assist: RAG needed to focus the LLM laser in large projects, along with Chain-of-Thought



Prompt Engineering



Lessons Learned: Taming LLM nondeterminism



To get consistent high-quality suggestions, you need to re-prompt (in the background), accumulate results shown to the user

Re-prompting not a waste

Newly-designed ranking to match LLM workflow (e.g., popularity of suggestions, heat map of the code affected by suggestions)

Sweet spot: tuning LLM hyperparameters (e.g., temperatures and number of iterations) is essential

- Higher randomness in Large Language Models is preferred when a solid validation framework exists

Executive Summary



IDE + LLM