Next-Generation Refactoring: Combining LLM Insights and IDE Capabilities for Extract Method

Dorin Pomian

Abhiram Bellur

Malinda Dilhara

Zarina Kurbatova

Andrey Sokolov

Egor Bogomolov

Timofey Bryksin

Danny Dig

Boulder

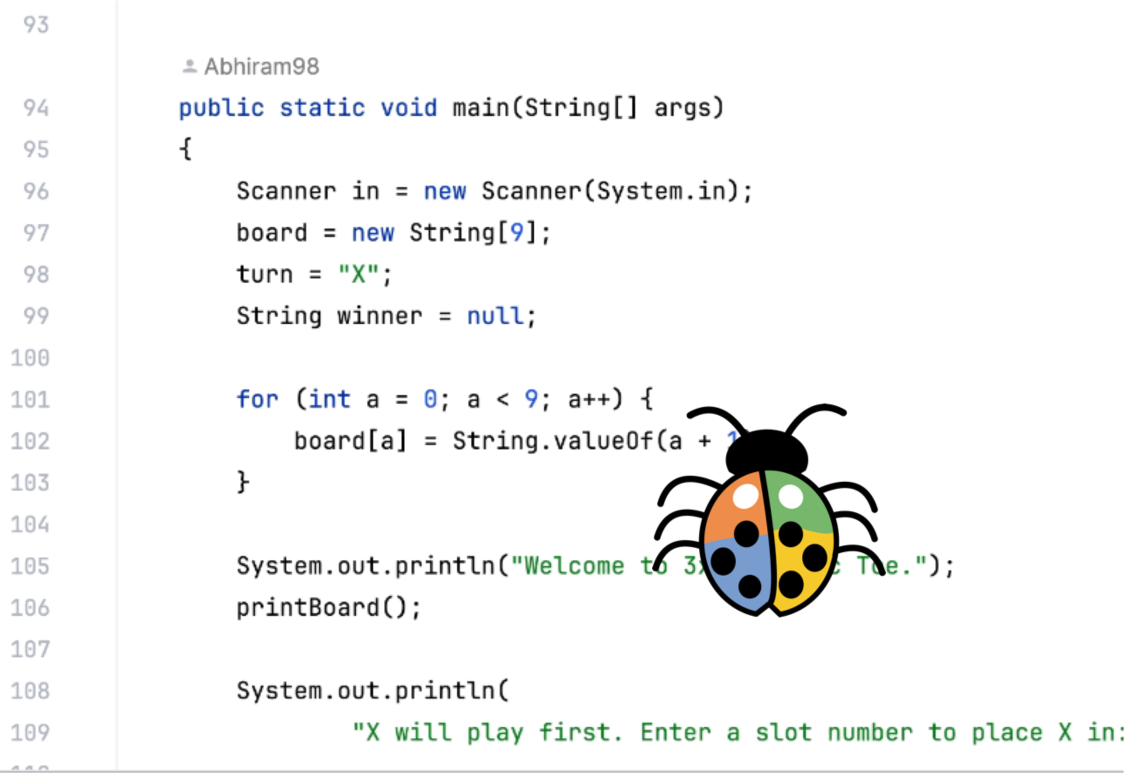
Boulder

Boulder

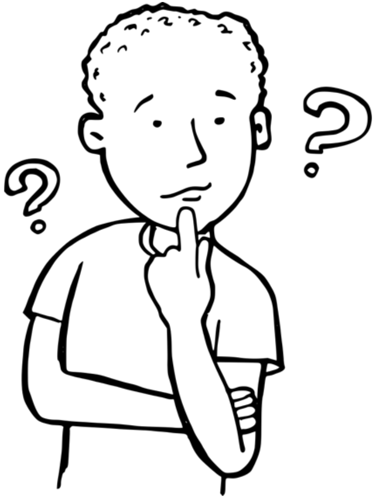
JETBRAINS Research

Boulder

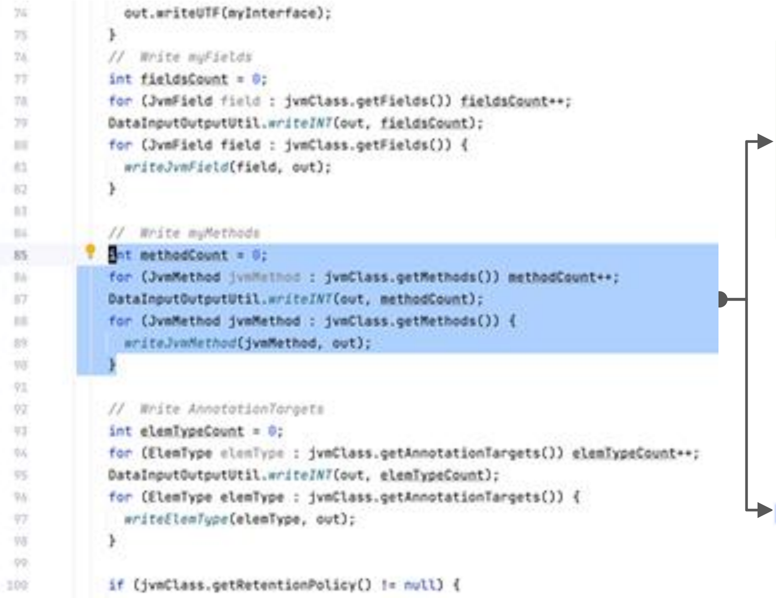
Long Methods In Codebases



```
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:
```

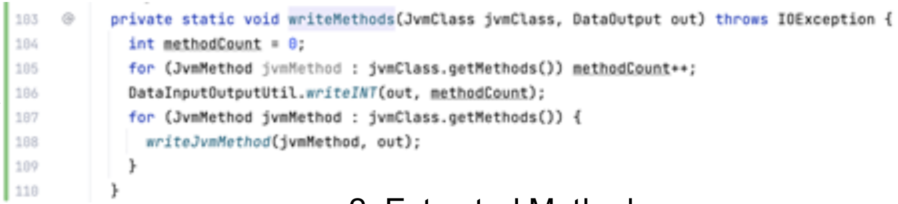


Extract Method Refactoring



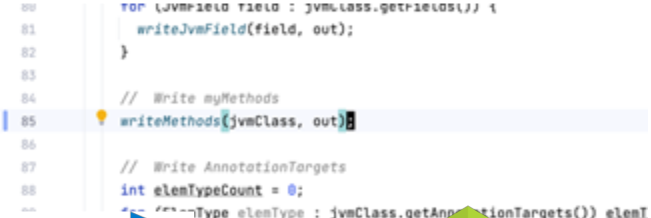
```
    out.writeUTF(myInterface);
  }
  //  Write myFields
  int fieldsCount = 0;
  for (JvmField field : jvmClass.getFields()) fieldsCount++;
  DataInputOutputUtil.writeINT(out, fieldsCount);
  for (JvmField field : jvmClass.getFields()) {
    writeJvmField(field, out);
  }

  //  Write myMethods
  int methodCount = 0;
  for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
  DataInputOutputUtil.writeINT(out, methodCount);
  for (JvmMethod jvmMethod : jvmClass.getMethods()) {
    writeJvmMethod(jvmMethod, out);
  }

  //  Write AnnotationTargets
  int elemTypeCount = 0;
  for (ElemType elemType : jvmClass.getAnnotationTargets()) elemTypeCount++;
  DataInputOutputUtil.writeINT(out, elemTypeCount);
  for (ElemType elemType : jvmClass.getAnnotationTargets()) {
    writeElemType(elemType, out);
  }

  if (jvmClass.getRetentionPolicy() != null) {
```

1. Original Method



```
private static void writeMethods(JvmClass jvmClass, DataOutput out) throws IOException {
  int methodCount = 0;
  for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
  DataInputOutputUtil.writeINT(out, methodCount);
  for (JvmMethod jvmMethod : jvmClass.getMethods()) {
    writeJvmMethod(jvmMethod, out);
  }
}
```

2. Extracted Method

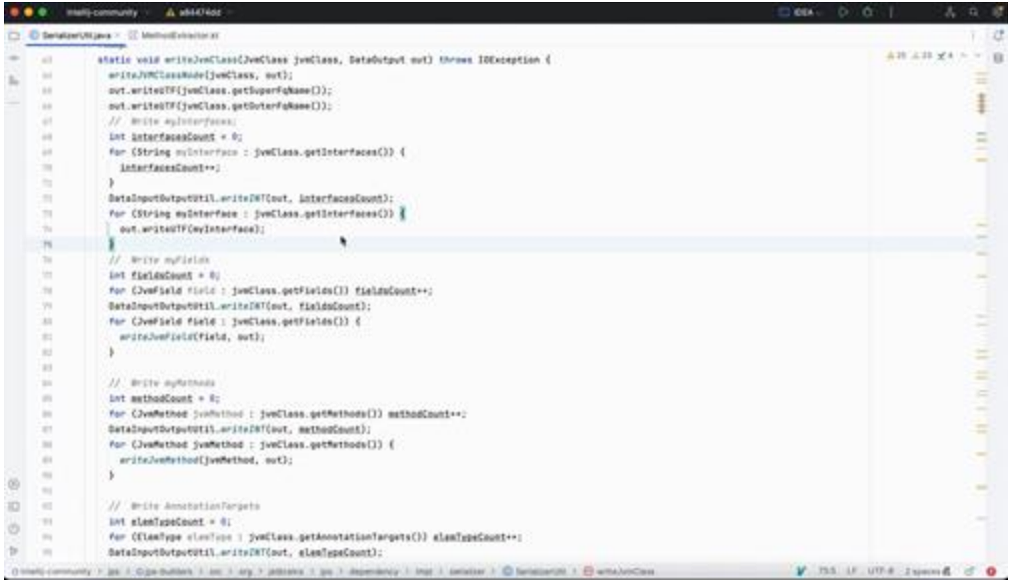


```
  for (JvmField field : jvmClass.getFields()) {
    writeJvmField(field, out);
  }

  //  Write myMethods
  writeMethods(jvmClass, out);

  //  Write AnnotationTargets
  int elemTypeCount = 0;
  for (ElemType elemType : jvmClass.getAnnotationTargets()) elemT
```

3. C

Current Extract Method Workflow in IntelliJ

- ✓ JetBrains’ IntelliJ IDEA has extract method capabilities
- ✗ Semi-automated process
- ✗ No automatic recommendations



```
static void writeJvmClass(JvmClass jvmClass, DataOutput out) throws IOException {
  writeJVMClassNode(jvmClass, out);
  out.writeUTF(jvmClass.getSuperFqName());
  out.writeUTF(jvmClass.getOuterFqName());
  //  Write myInterfaces;
  int interfacesCount = 0;
  for (String myInterface : jvmClass.getInterfaces()) {
    interfacesCount++;
  }
  DataInputOutputUtil.writeINT(out, interfacesCount);
  for (String myInterface : jvmClass.getInterfaces()) {
    out.writeUTF(myInterface);
  }
  //  Write myFields
  int fieldsCount = 0;
  for (JvmField field : jvmClass.getFields()) fieldsCount++;
  DataInputOutputUtil.writeINT(out, fieldsCount);
  for (JvmField field : jvmClass.getFields()) {
    writeJvmField(field, out);
  }

  //  Write myMethods
  int methodCount = 0;
  for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
  DataInputOutputUtil.writeINT(out, methodCount);
  for (JvmMethod jvmMethod : jvmClass.getMethods()) {
    writeJvmMethod(jvmMethod, out);
  }

  //  Write AnnotationTargets
  int elemTypeCount = 0;
  for (ElemType elemType : jvmClass.getAnnotationTargets()) elemTypeCount++;
  DataInputOutputUtil.writeINT(out, elemTypeCount);
```

Extract Method Research

✓ Many research tools for recommending fragments to extract

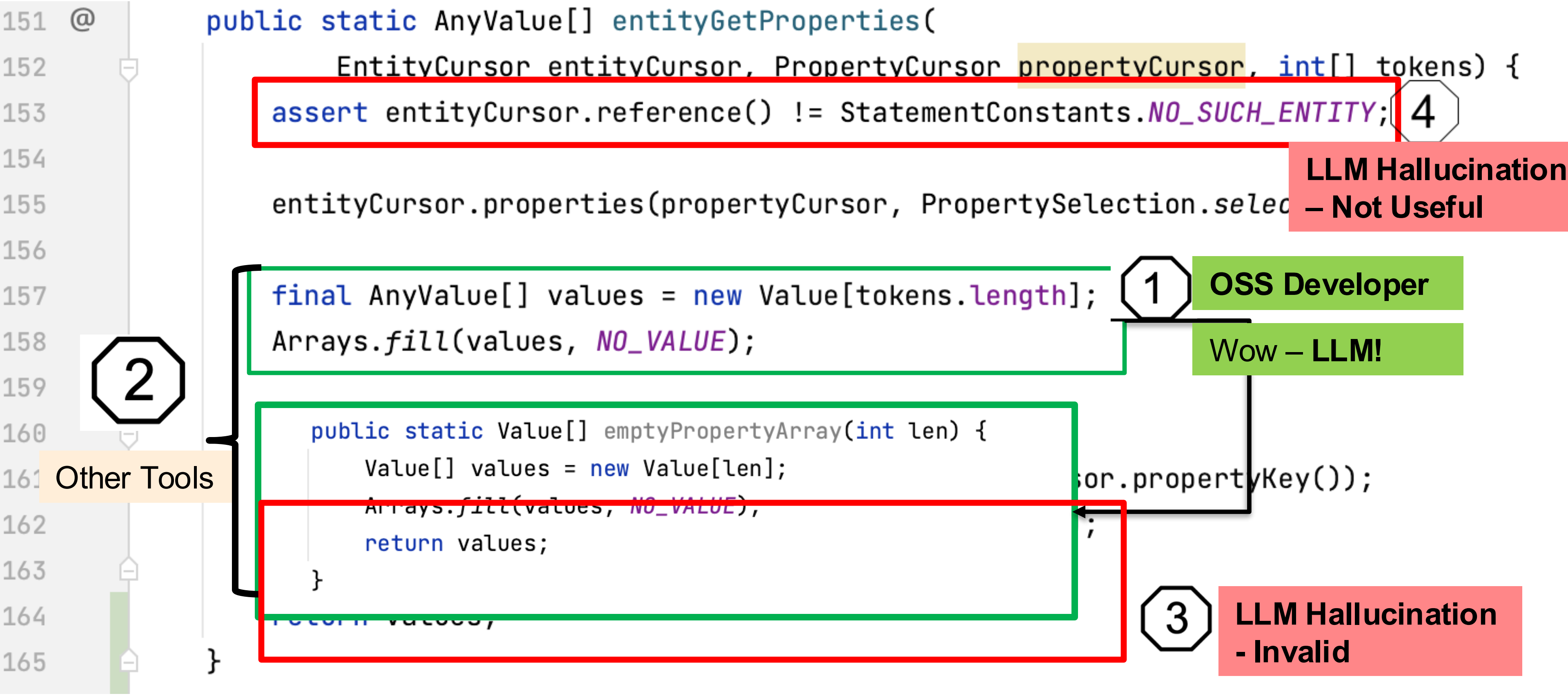
- *JDeodorant*
- *JExtract*
- *LiveREF*
- *REMS*
- *GEMS*
- *SEMI*

? Optimize software quality metrics

✗ Generate refactorings that do not align with developers’ preferences

Motivating Example from Open-Source Project (Neo4j)

a05a8c5



```
public static AnyValue[] entityGetProperties(
        EntityCursor entityCursor, PropertyCursor propertyCursor, int[] tokens) {
    assert entityCursor.reference() != StatementConstants.NO_SUCH_ENTITY;

    entityCursor.properties(propertyCursor, PropertySelection.selec

    final AnyValue[] values = new Value[tokens.length];
    Arrays.fill(values, NO_VALUE);

                                                   sor.propertyKey());
                                                   ;

    return values;
}
```

```
public static Value[] emptyPropertyArray(int len) {
    Value[] values = new Value[len];
    Arrays.fill(values, NO_VALUE);
    return values;
}
```

LLM Hallucination – Not Useful

OSS Developer

Wow – LLM!

Other Tools

LLM Hallucination - Invalid

Our solution: *LLM + IDE Static Analysis*

- Leverage creative capabilities of LLMs

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

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

- IntelliJ IDEA plugin implementation – EM-Assist

EM-Assist Workflow

Empirical Evaluation

Datasets:

- Synthetic Corpus of 122 scenarios
- Mined 1752 real-world Extract-Method refactorings from OSS

RQ1: How effective is Vanilla LLM?

RQ2: What LLM hyper-parameters work best?

RQ3: How effective is EM-Assist?

RQ4: How useful is EM-Assist?

RQ1: How effective is Vanilla LLM?

Asked an LLM to replicate an oracle of refactoring situations

✓ LLMs are creative and prolific: averaging 27 suggestions per method

✗ **Hallucination** Invalid - 44.4% of the suggestions are invalid, resulting in non-compiling code, or semantically not equivalent

✗ **Hallucination** Not Useful - 14.8% of suggestions are not useful (e.g. one liners, or entire method body)

RQ3: How effective is EM-Assist

Oracle of actual 1752 extract method refactorings from OSS

- EM-Assist achieved 53.4% recall rate
- Compared to 39.4% recall rate by JExtract (best in class using static analysis)

EM-Assist better aligns with how expert developers performed refactorings in the wild.

RQ4: How useful is EM-Assist?

Fire-house survey – New commits

16 expert developers participated in the survey, with 81.3% giving a positive rating

“It looks super cool so far! :fire:”

“Thank you for interesting suggestions! Hope to see this in production in the future.”

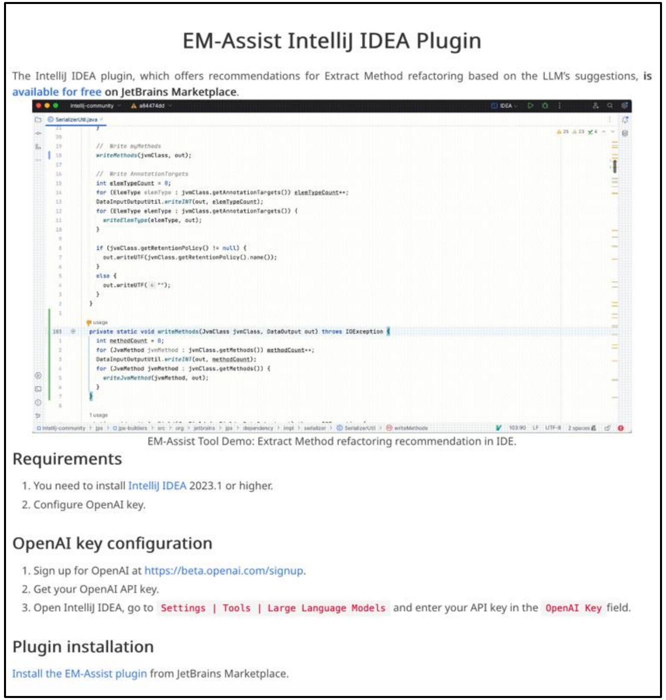
“These suggestions made me look at this code with new eyes, and I will try to refactor it”

Executive Summary

- LLM’s create a wow effects,
  but also have a **high hallucination rate – 59.2%**
- Tame Hallucinations
- Tame the non-determinism
- EM-Assist outperforms previous state-of-the-art and aligns with how expert developers perform refactoring.

IDE + LLM + Human >> Sum of the individual parts

Ongoing work: Moving beyond one refactoring at a time
⇒ Refactoring Plan executed by an Agent



EM-Assist IntelliJ IDEA Plugin

The IntelliJ IDEA plugin, which offers recommendations for Extract Method refactoring based on the LLM’s suggestions, **is available for free on JetBrains Marketplace.**

```
    }

    // Write myMethods
    writeMethods(jvmClass, out);

    // Write AnnotationTargets
    int elemTypeCount = 0;
    for (ElemType elemType : jvmClass.getAnnotationTargets()) elemTypeCount++;
    DataInputOutputUtil.writeINT(out, elemTypeCount);
    for (ElemType elemType : jvmClass.getAnnotationTargets()) {
      writeElemType(elemType, out);
    }

    if (jvmClass.getRetentionPolicy() != null) {
      out.writeUTF(jvmClass.getRetentionPolicy().name());
    }
    else {
      out.writeUTF( s: "");
    }
  }

  usage
  private static void writeMethods(JvmClass jvmClass, DataOutput out) throws IOException {
    int methodCount = 0;
    for (JvmMethod jvmMethod : jvmClass.getMethods()) methodCount++;
    DataInputOutputUtil.writeINT(out, methodCount);
    for (JvmMethod jvmMethod : jvmClass.getMethods()) {
      writeJvmMethod(jvmMethod, out);
    }
  }

  1 usage
```

intellij-community › jps › jps-builders › src › org › jetbrains › jps › dependency › impl › serializer › SerializerUtil › writeMethods
103:90 LF UTF-8 2 spaces

EM-Assist Tool Demo: Extract Method refactoring recommendation in IDE.

Requirements

1. You need to install IntelliJ IDEA 2023.1 or higher.
2. Configure OpenAI key.

OpenAI key configuration

1. Sign up for OpenAI at https://beta.openai.com/signup.
2. Get your OpenAI API key.
3. Open IntelliJ IDEA, go to `Settings | Tools | Large Language Models` and enter your API key in the `OpenAI Key` field.

Plugin installation

Install the EM-Assist plugin from JetBrains Marketplace.

Tool Datasets

Demo

Your questions: **Suggestion Quality, Ranking, and Human Factors**

How does the system reconcile objective software engineering best practices with subjective developer preferences, particularly when its definition of "usefulness" risks reinforcing existing patterns at the expense of novel solutions?

Does the ranking mechanism inherently down-rank rare but superior refactorings, and have alternative presentations—like grouping by intent or removing ranks entirely—been explored to counter this bias and better empower developers?

Ensuring Correctness and Handling Hallucinations

- Beyond the limits of static analysis, how does the system guarantee that a refactoring suggestion is semantically equivalent and preserves control flow, thereby catching subtle hallucinations?

- How does it differentiate these errors from intentional, beneficial logic changes a developer might want, and what are the performance and feasibility trade-offs of integrating a deeper analysis tool, such as a language server, into the validation pipeline?

LLM Behavior, Prompting, and Configuration

- What explains the counterintuitive result of GPT-3.5 outperforming GPT-4 for this task, and how sensitive is the system to future model updates?

- In controlling the output, what are the trade-offs between constraining the model via prompt engineering versus applying post-processing filters?

- How is the model's non-determinism managed, and could it be leveraged as a feature—using higher temperatures to generate more diverse, semantically equivalent suggestions for the developer to choose from?

Scalability, Generalization, and Long-Term Impact

- How does this tool's performance scale to massive enterprise codebases with significant legacy code?

- What is its measurable long-term impact on code maintainability and potential runtime performance?

- What strategies can make the associated LLM costs economically viable for smaller businesses?

System Architecture and Adaptation

- From an architectural standpoint, how does the IDE integration manage asynchronous calls to maintain UI responsiveness, and could different IDE versions improve performance?

- Can the system adapt beyond one-shot suggestions to learn and enforce project-specific coding conventions by retaining context over time?

- How do its heuristics, such as the 88% filter, balance the goal of eliminating trivial suggestions against the risk of accidentally discarding valid, large-scale refactorings?

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

Malinda Dilhara

University of Colorado Boulder

Abhiram Bellur

University of Colorado Boulder

Timofey Bryksin

JET BRAINS Research

Danny Dig

University of Colorado Boulder

JET BRAINS Research

Motivation

Developers repeat code changes

Repeated changes happen because:

- Adoption of shared coding idioms
- Adherence to common best practices
- Tackle similar programming challenges
  - Fixing bugs, API updates, Language upgrades

Code change pattern (CPAT)

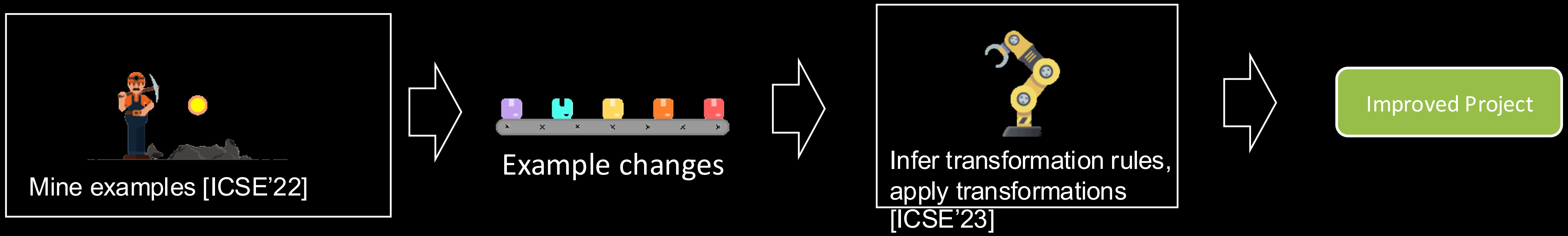
```
number = 0
for x in int_list:
  number = number + x
```

➡

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

Commit c8b28432 in GitHub project NifTK/NiftyNet

Overview of Transformation by Example (TBE)



Mine examples [ICSE'22]

Example changes

Infer transformation rules, apply transformations [ICSE'23]

Improved Project

There are many other Transformation by Example(TBE) systems

PyEvolve (ICSE-2023)

PyEvolve is a TBE system

Code example

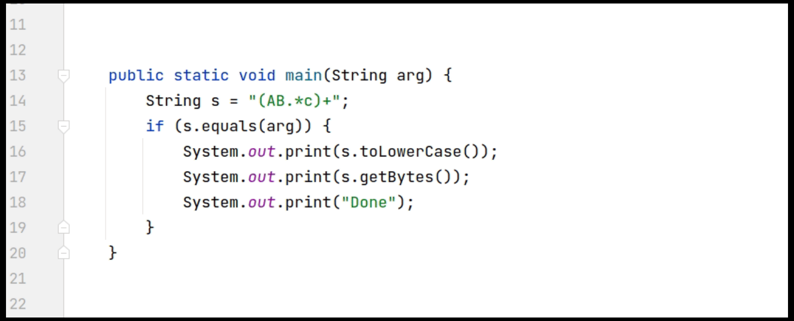
```
- new SqlScriptExecutor("foo",() => true , null)
+ new SqlScriptExecutor("foo",() => true, null, () => Substitute.For<IJournal>())
```

Transformation Rule

new SqlScriptExecutor(X_1,X_2,X_3)

→ new SqlScriptExecutor($X1$,$X2$,$X3$, () => Substitute.For<IJournal>())

APIFix - Gao et al. (OOPSLA - 2021)



```
public static void main(String arg) {
    String s = "(AB.*c)+";
    if (s.equals(arg)) {
        System.out.print(s.toLowerCase());
        System.out.print(s.getBytes());
        System.out.print("Done");
    }
}
```

TCInfer – Ketkar et al. (ICSE-2022)
collaboration ML4SE @ JetBrains Research

Different syntactic variants are challenging

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

➡

```
number= np.sum(int_list)
```

```
count = 0
for i in range(len(int_list)):
      count += int_list[i]
```

Different syntactic variants are challenging

```
count = 0
i= 0
while i < len (int_list):
      count += int_list[i]
      i++
```

```
result = 0
for i in range(len(int_list)):
      result = result + int_list[i]
```

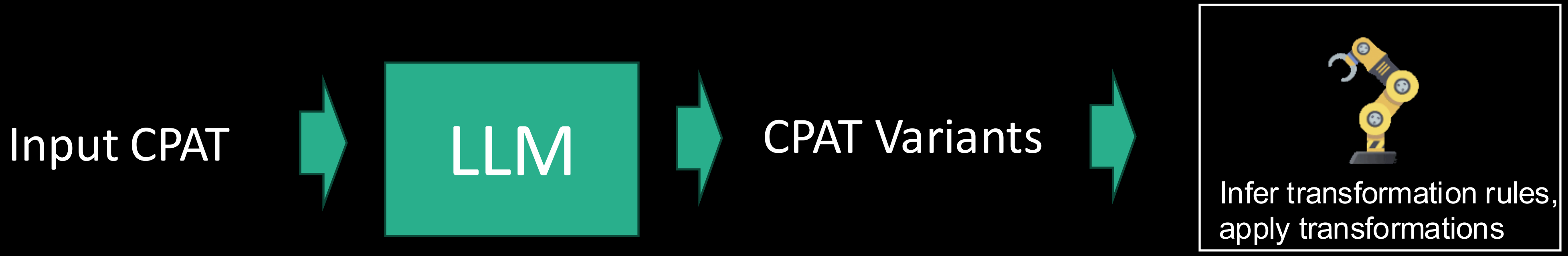
```
count = 0
for i in range(len(int_list)):
   count += int_list
```

```
result = 0
for index, value in enumerate(int list):
      result = result + value
```

```
result = 0
for index, value in enumerate (int_list):
      result += value
```

Key Idea

- LLMs’ training data comprises of various ways that developers write the same piece of code.
- Use LLMs to generate different variations of the input code



Input CPAT

LLM

CPAT Variants

Infer transformation rules, apply transformations

LLMs Hallucinate

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

→

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

Original CPAT

Incorrect

```
count = 0
for i in range(len(int_list)):
  count += 5
```

Unrealistic

```
count = 0
for i in sorted(int_list):
  count += i
```

Not applicable

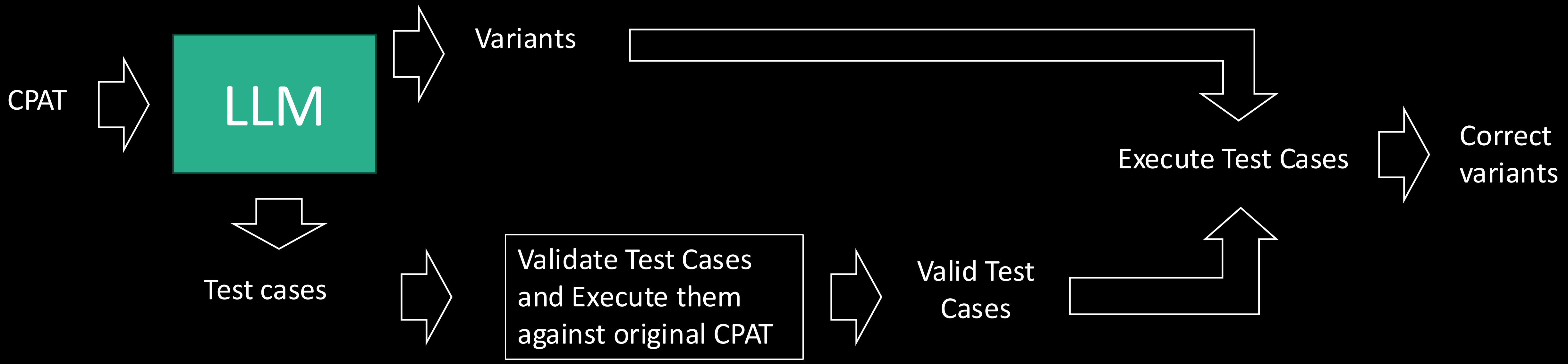
```
count = sum(int_list)
```

LLM + Static Code Analysis + Dynamic Code Analysis

Selecting correct Variants

```
count = 0
for i in range(len(int_list)):
    count += 5
```

- Generate Test Cases
- Execute Test Cases Select Variants Semantically Equivalent to the CPAT



Selecting Realistic Variants

```
count = 0
for i in sorted(int_list):
  count += int_list[i]
```

Detecting all the unrealistic variants is hard

The goal is to reduce as many unrealistic variants as possible.

Fine-tune the randomness to generate fewer unrealistic variants

Selecting Applicable Variants

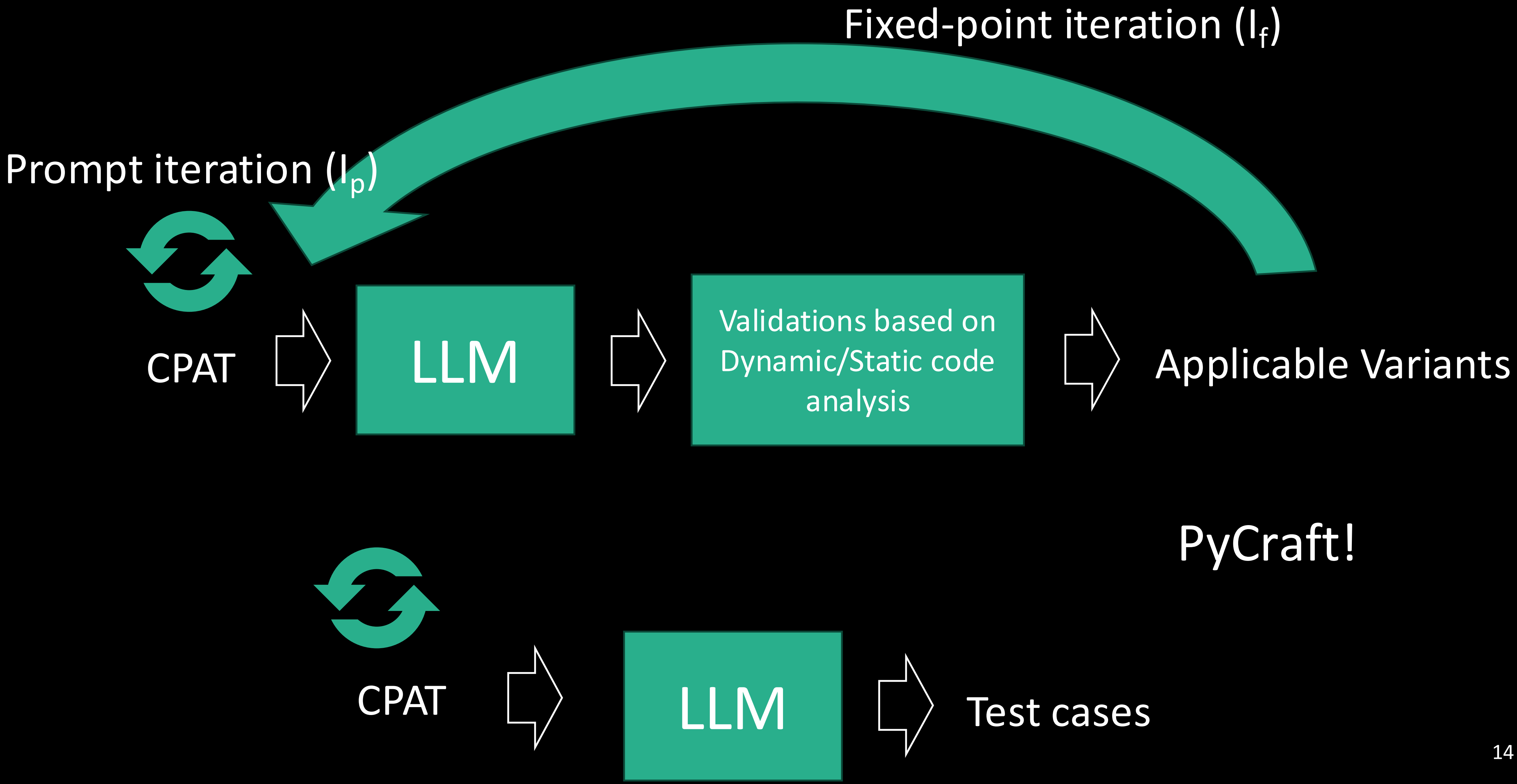
~~count = sum(int_list)~~

Use static analysis-based rules to remove not applicable variants

1) Number of control nodes (e.g., For loops) are equal in both original CPAT and Variant?

2) Does the variants contain any new declarations (e.g., method declarations)

Harnessing full power of LLMs



Fixed-point iteration (I_f)

Prompt iteration (I_p)

CPAT

LLM

Validations based on Dynamic/Static code analysis

Applicable Variants

PyCraft!

CPAT

LLM

Test cases

Evaluation

RQ1) How effective are LLMs at generating variations?

RQ2) How effective are LLMs at generating test-cases?

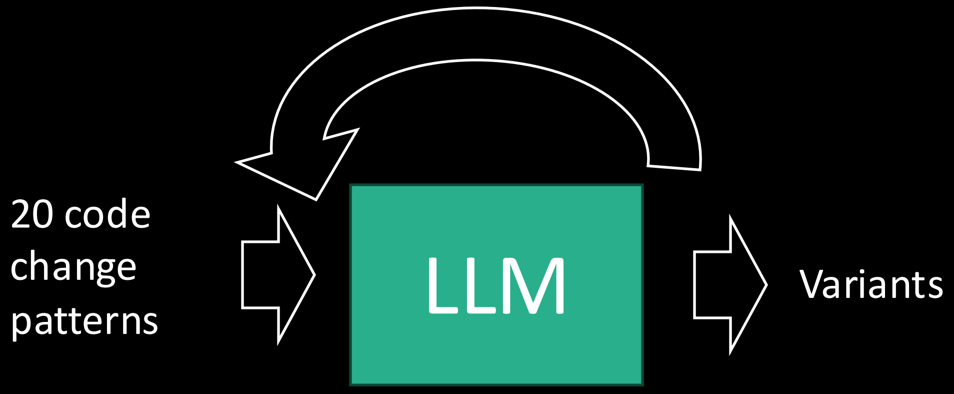
RQ3) What are the optimal parameters for generating unseen variants?

RQ4) What are the optimal parameters for generating test cases?

RQ5) How effective is PyCraft at finding new opportunities and performing transformations over the previous state-of-the-art?

RQ6) How useful are the generated program transformations?

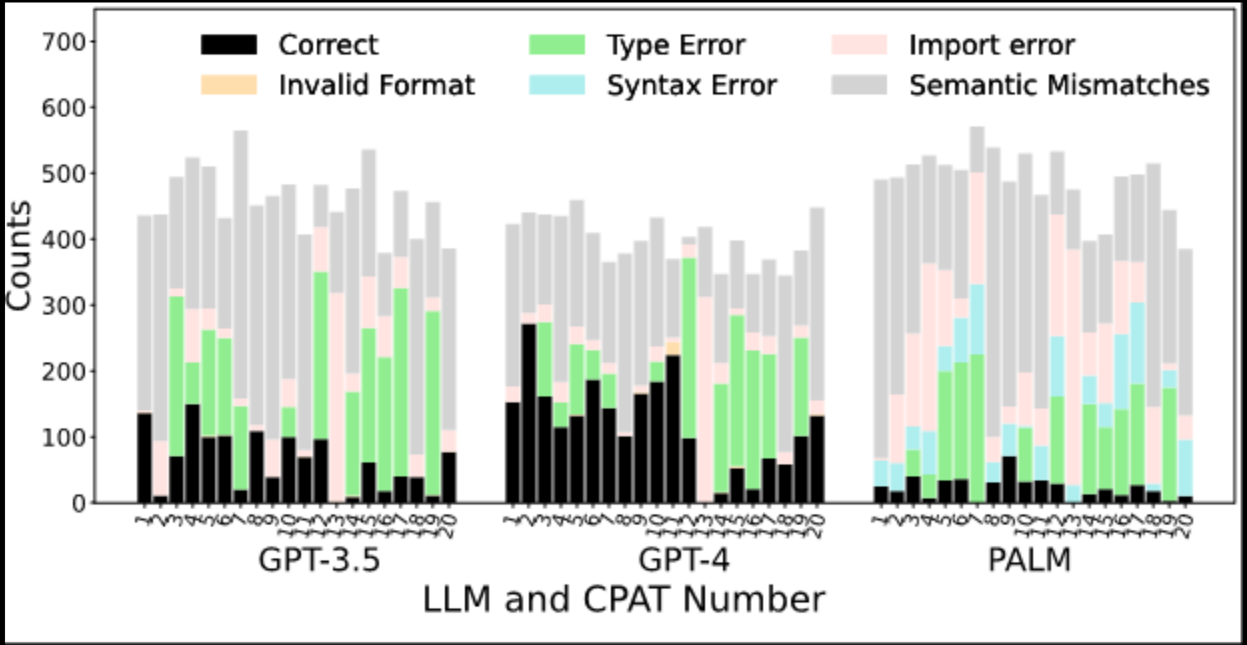
RQ1). How effective are LLMs at generating variations?



20 code change patterns

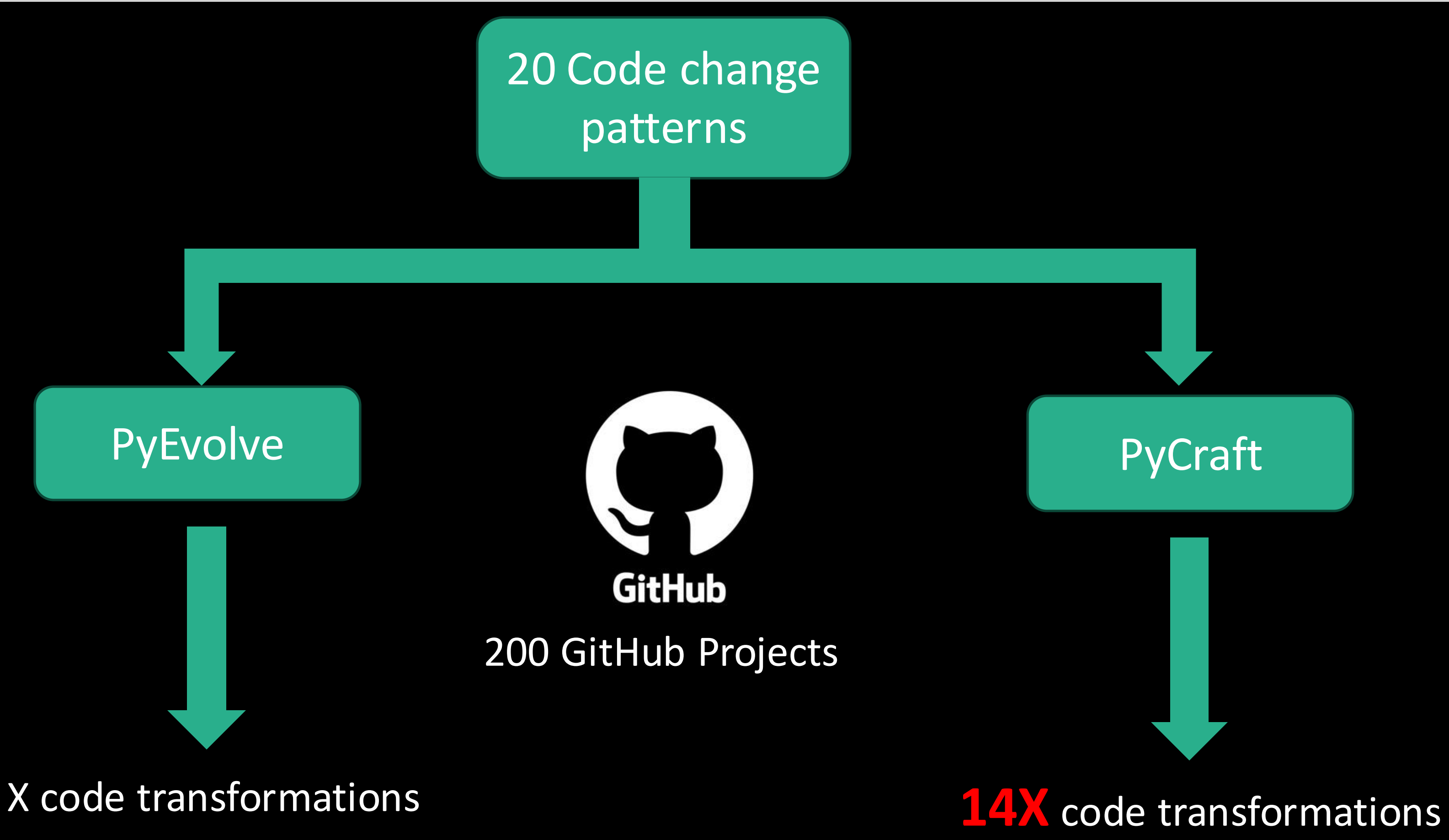
LLM

Variants



LLMs excel in generating unseen variants (584 per CPAT) but also produce errors (65%).

RQ5). How effective is PyCraft at finding new opportunities and performing transformations over the baseline?



20 Code change patterns

PyEvolve

GitHub

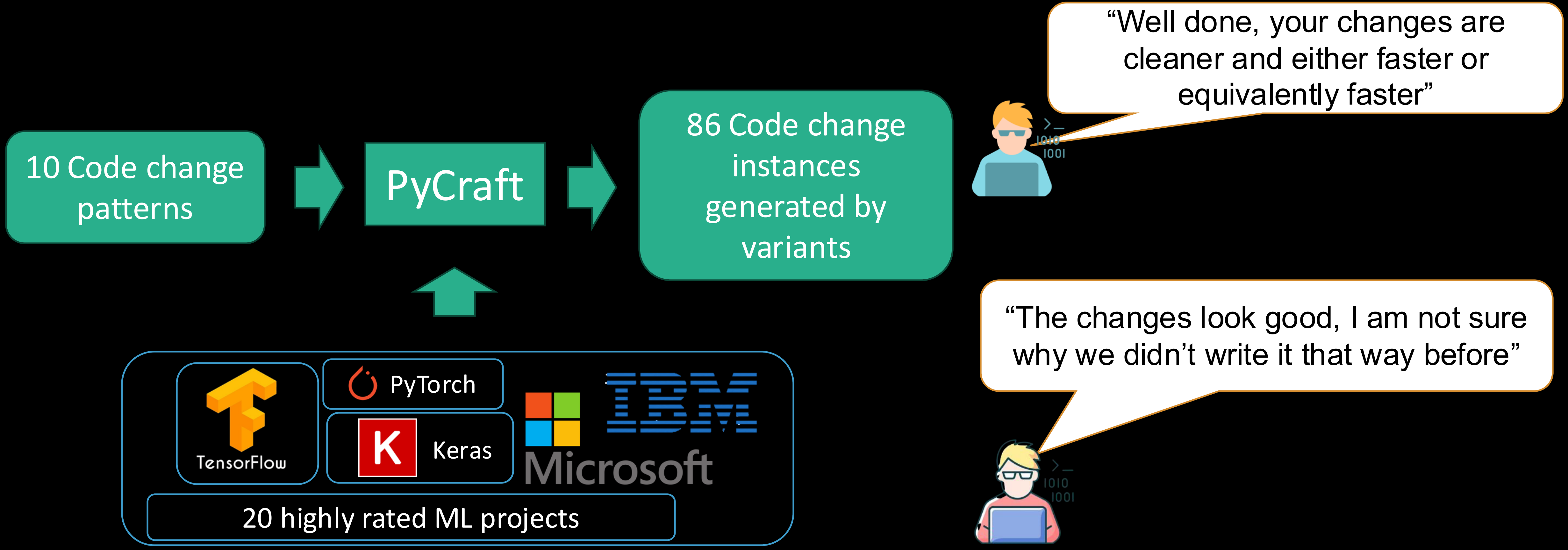
200 GitHub Projects

PyCraft

X code transformations

14X code transformations

RQ6). How useful are the generated program transformations?



10 Code change patterns

PyCraft

86 Code change instances generated by variants

20 highly rated ML projects

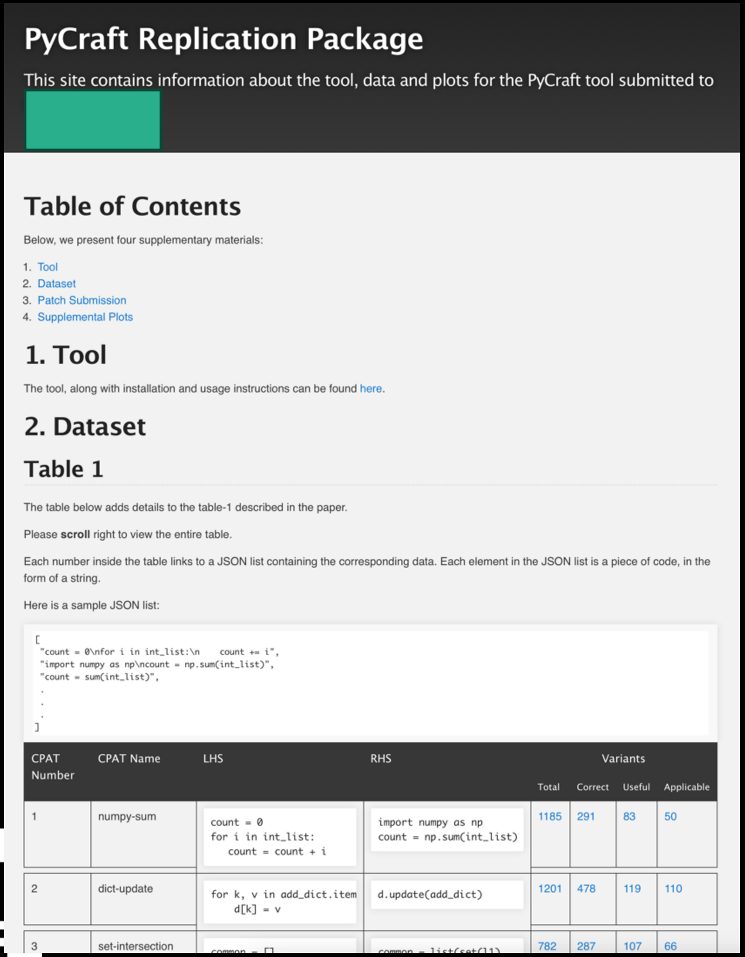
"Well done, your changes are cleaner and either faster or equivalently faster"

"The changes look good, I am not sure why we didn't write it that way before"

PyCraft generated 86 code patches, of which developers accepted 72 (83%)

Summary

- PyCraft is a novel transformation-by-example system that is 14x better than previous state-of-the-art.
- Harness LLM power, to generate previously unseen variants (58 per CPAT)
- Develop novel techniques to filter the high rate of LLM hallucinations
- Discovered best-practices to get the most performance out of LLM
- Submitted 86 patches to 20 open-source projects, of which developers accepted 72 (83%)



PyCraft Replication Package

This site contains information about the tool, data and plots for the PyCraft tool submitted to [redacted]

Table of Contents

Below, we present four supplementary materials:

1. Tool

The tool, along with installation and usage instructions can be found here.

2. Dataset

Table 1

The table below adds details to the table-1 described in the paper.

Please **scroll** right to view the entire table.

Each number inside the table links to a JSON list containing the corresponding data. Each element in the JSON list is a piece of code, in the form of a string.

Here is a sample JSON list:

```
[
 "count = 0\nfor i in int_list:\n    count += i",
 "import numpy as np\ncount = np.sum(int_list)",
 "count = sum(int_list)",
 .
 .
 .
]
```

CPAT Number	CPAT Name	LHS	RHS	Variants			
				Total	Correct	Useful	Applicable

1	numpy-sum	`count = 0`
		`for i in int_list:`
		`   count = count + i`
			`import numpy as np`
			`count = np.sum(int_list)`
				1185	291	83	50

2	dict-update	`for k, v in add_dict.item`
		`    d[k] = v`
			`d.update(add_dict)`	1201	478	119	110

3	set-intersection	`common = []`	`common = list(set(l1)`	782	287	107	66

Dataset and Tool

How can you use our tool to your own research?

Generalizability, Robustness, and Future-Proofing

- How does PyCraft ensure long-term robustness and adaptability?
- How can it remain effective as underlying LLMs evolve, and can its hyperparameters be auto-tuned for new models?
- How would its validation process adapt to languages with complex constraints like Rust's borrow checker?
- How stable are the inferred rules across runs?
- How critical are "unseen" variants to keeping pace with software evolution?

Integration into Developer Workflow & Real-World Application

- How does PyCraft integrate into a real-world developer workflow, combining refactoring with new feature development?

- How does it make context-aware decisions (e.g., CPU vs. GPU targets), support legacy system modernization?

- How does it embody a nuanced refactoring philosophy that distinguishes between code needing abstraction and code that is intentionally verbose for clarity?

Evaluation, Metrics, and Limitations

- Beyond immediate usefulness, how can the long-term impact on code maintainability be measured?
- What is the system's false negative rate for good edits?
- What are the primary limitations where the rule-mining breaks, such as with multi-location or non-behavior-preserving changes?
- Crucially, how much of the performance uplift is attributable specifically to the LLM's generative diversity versus the underlying search and abstraction framework?

Security Considerations

What security considerations govern the use of "unseen" LLM-generated variants for creating automated transformation rules?

What specific safeguards are in place to prevent the introduction of vulnerabilities or subtle bugs through these novel, unvetted code patterns?

Core Mechanism & Implementation

- What is PyCraft's operational pipeline?
- What complexity of Code Patterns and Transformations (CPATs)—from one-liners to larger blocks—can it handle?
- How are its prompts programmatically architected from examples?
- What search mechanism efficiently finds all CPAT instances in large codebases, and how does pattern complexity impact that search?

The Role and Application of LLMs

- What is the precise relationship between the few-shot learning, in-context learning, and prompt engineering methodologies used?
- What specific prompts enable the LLM to perform auxiliary tasks like type inference and test generation?
- Furthermore, how generalizable are the chosen hyperparameters (e.g., Temperature) across different domains, and what methods beyond tuning are used to mitigate hallucinations?

Managing and Validating Generated Code Variants

- How does PyCraft manage the variant lifecycle to ensure quality and sufficiency? How does it prevent duplicate variants and verify new ones are meaningfully distinct?

- Is there an automated process for filtering "non-useful” variants?

- To overcome manual validation, how can the "usefulness" judgment be scaled or automated, perhaps via LLMs or by mining code review signals?

- Given diminishing returns, how does the system determine when to stop generating variants?

Scalability, Cost, and Performance

- How do PyCraft's performance and cost—including computation, latency, and finances—scale with codebase size and CPAT complexity?
- What are the practical upper and lower bounds for a codebase where the tool is effective?
- How does its latency compare to traditional refactoring tools?
- Is the overall re-prompting approach economically viable for large-scale enterprise use?