Towards Reliable Machine Learning Models for Code

Foutse Khomh, PhD, Ing.
foutse.khomh@polymtl.ca
@SWATLab

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Mila

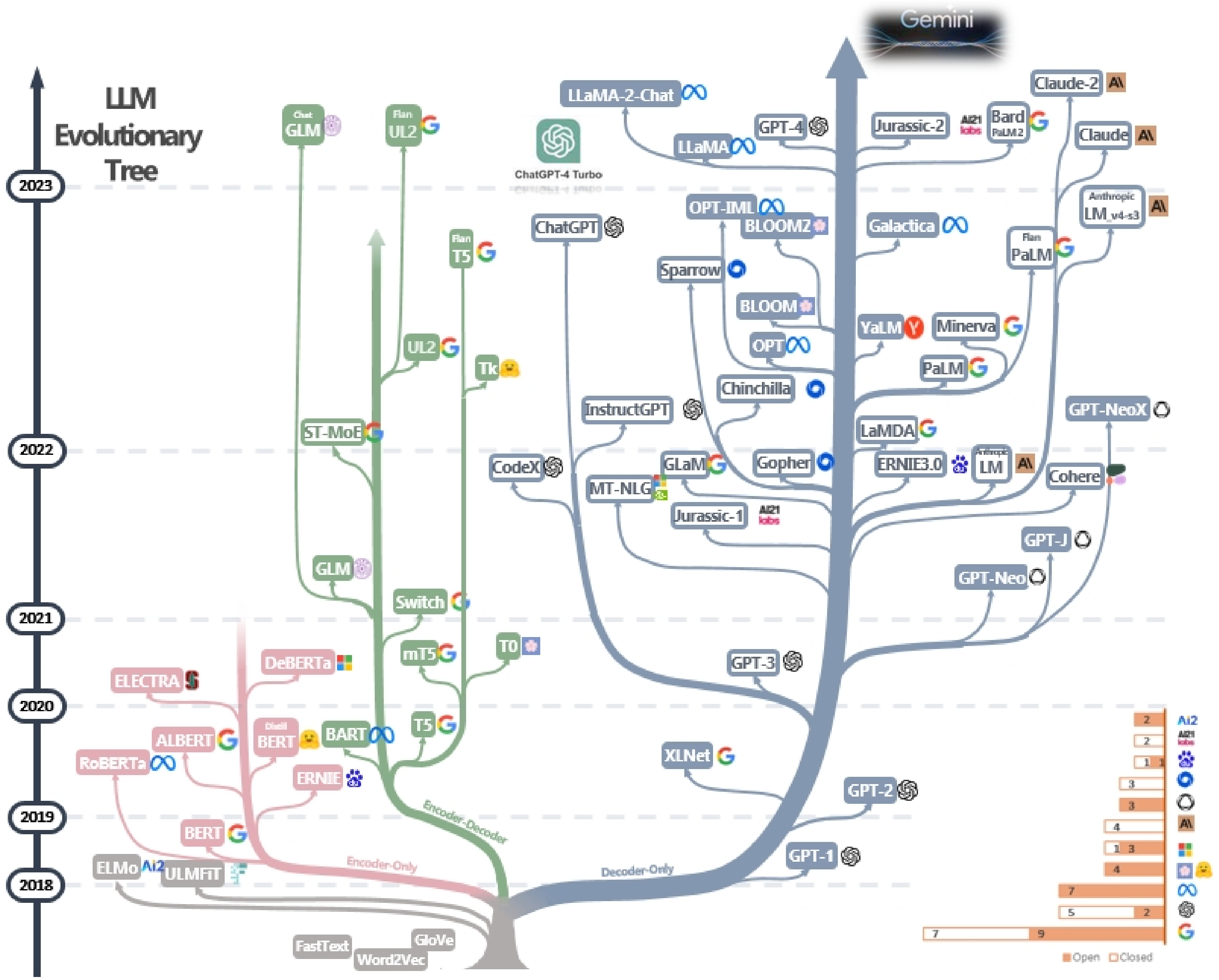
SEMLA
Trustworthy Engineering of AI Software

S.W.A.T
SoftWare Analytics and Technologies Lab

Some Team Members

We are entering in an Era of AI-assisted Software Engineering

The LLM revolution



Large Language Models (LLM) are increasingly being deployed to solve complex SE tasks!

Gemini

GitHub Copilot

CLAUDE 2

GPT-4

Code Llama 2

HUAWEI PanGu-Coder 2

AlphaCode

CodeT5

StarCoder2 built by BigCode

CODEGEN

#	Model	pass@1
1	🥇 GPT-4 (May 2023) ✨	88.4
2	🥈 GPT-4-Turbo (Nov 2023) ✨	85.4
3	🥉 claude-3-opus (Mar 2024) ✨	82.9
4	DeepSeek-Coder-33B-instruct ✨	81.1
5	WizardCoder-33B-V1.1 ✨	79.9
6	OpenCodeInterpreter-DS-33B ✨ 💙	79.3
7	OpenCodeInterpreter-DS-6.7B ✨ 💙	77.4
8	speechless-codellama-34B-v2.0 ✨ 💙	77.4
9	GPT-3.5-Turbo (Nov 2023) ✨	76.8
10	Magicoder-S-DS-6.7B ✨ 💙	76.8
11	XwinCoder-34B ✨	75.6
12	DeepSeek-Coder-7B-instruct-v1.5 ✨	75
13	code-millenials-34B ✨	74.4
14	DeepSeek-Coder-6.7B-instruct ✨	73.8
15	GPT-3.5 (May 2023) ✨	73.2

Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation

Jiawei Liu* **Chunqiu Steven Xia*** **Yuyao Wang** **Lingming Zhang**

University of Illinois Urbana-Champaign Nanjing University

{jiawei6, chunqiu2, lingming}@illinois.edu yuyao6@outlook.com

ClassEval: A Manually-Crafted Benchmark for Evaluating LLMs on Class-level Code Generation

Xueying Du Mingwei Liu Kaixin Wang Hanlin Wang Junwei Liu
Yixuan Chen Jiayi Feng Chaofeng Sha Xin Peng Yiling Lou
Fudan University
Shanghai, China
{xueyingdu21,kxwang23,wanghanlin23}@m.fudan.edu.cn
{jwliu22,23212010005,23210240148}@m.fudan.edu.cn
{liumingwei,cfsha,pengxin,yilinglou}@fudan.edu.cn

SWE-BENCH: CAN LANGUAGE MODELS RESOLVE REAL-WORLD GITHUB ISSUES?

Carlos E. Jimenez*[1,2] **John Yang*[1,2]** **Alexander Wettig[1,2]**
Shunyu Yao[1,2] **Kexin Pei[3]** **Ofir Press[1,2]** **Karthik Narasimhan[1,2]**
[1]Princeton University [2]Princeton Language and Intelligence [3]University of Chicago

resources.github.com/learn/pathways/copilot/essentials/essentials-of-github-copilot/

Get started with GitHub Copilot >

Industry expert insight from:

ASOS: ASOS is a destination for fashion loving 20-somethings, with more than 23M active customers in over 200 countries worldwide. Through its leading web and app experiences, customers can shop from close to 900 partner brands and ASOS's selection of fashion-led own-brand labels. ASOS shares their self-serve approach to GitHub Copilot, which empowers engineers to take advantage of its features with minimal toil.

CARIAD, a Volkswagen Group company: CARIAD is building software to make automotive mobility safer, more sustainable, and more comfortable in a new way. They use GitHub Copilot to boost productivity, streamline development processes, enhance code quality, and accelerate project timelines. This module will explore how CARIAD integrates GitHub Copilot into their daily workflows, ensuring a seamless and efficient development experience.

Shopify: Shopify is a provider of essential internet infrastructure for commerce. Shopify makes commerce better for everyone with a platform and services that are engineered for reliability, while delivering a better shopping experience for consumers everywhere. Shopify shares how their engineering leaders strategically evangelized GitHub Copilot adoption internally to achieve a 90%+ adoption rate with more than 24,000 lines of code accepted everyday.

90% + adoption rate with more than 24,000 lines /day

Productivity Assessment of Neural Code Completion

Albert Ziegler
wunderalbert@github.com
GitHub, Inc.
San Francisco, USA

Eirini Kalliamvakou
ikaliam@github.com
GitHub, Inc.
San Francisco, USA

X. Alice Li
xalili@github.com
GitHub, Inc.
San Francisco, USA

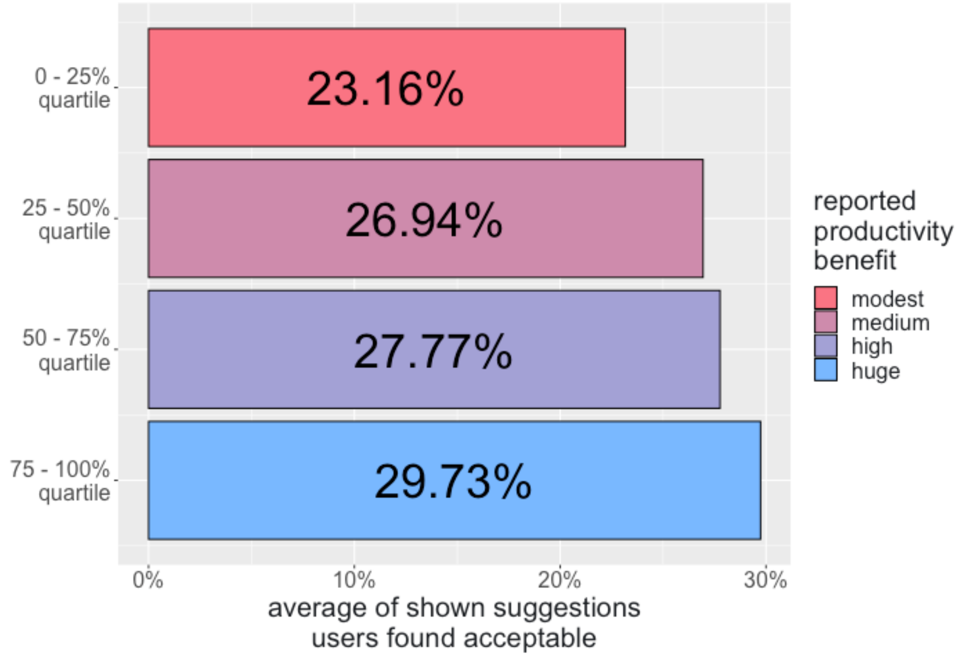
Andrew Rice
acr31@github.com
GitHub, Inc.
San Francisco, USA

Devon Rifkin
drifkin@github.com
GitHub, Inc.
San Francisco, USA

Shawn Simister
narphorium@github.com
GitHub, Inc.
San Francisco, USA

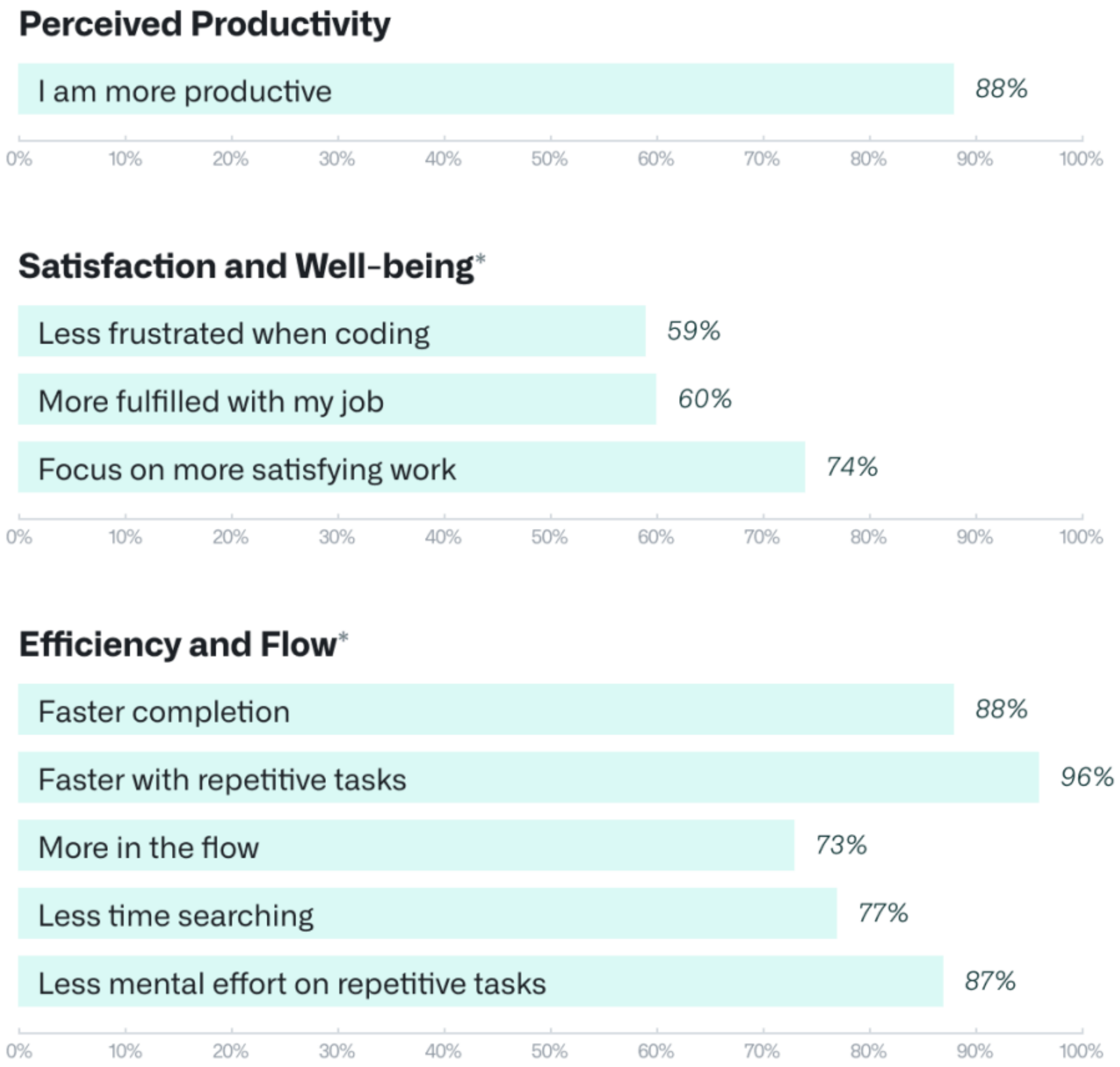
Ganesh Sittampalam
hsenag@github.com
GitHub, Inc.
San Francisco, USA

Edward Aftandilian
eaftan@github.com
GitHub, Inc.
San Francisco, USA



GitHub Copilot is behind an average of **46% of a developers' code across all programming languages—and in Java, that number jumps to 61%.**

When using GitHub Copilot...



https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

arXiv:2206.15331v2 [cs.SE] 14 Apr 2023

GitHub Copilot AI pair programmer: Asset or Liability?

Arghavan Moradi Dakhel*, Vahid Majdinasab*, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais

Polytechnique Montreal, Montreal, Canada

Zhen Ming (Jack) Jiang

York University, Toronto, Canada

Abstract

Automatic program synthesis is a long-lasting dream in software engineering. Recently, a promising Deep Learning (DL) based solution, called Copilot, has been proposed by OpenAI and Microsoft as an industrial product. Although some studies evaluate the correctness of Copilot solutions and report its issues, more empirical evaluations are necessary to understand how developers can benefit from it effectively. In this paper, we study the capabilities of Copilot in two different programming tasks: (i) generating (and reproducing) correct and efficient solutions for fundamental algorithmic problems, and (ii) comparing Copilot's proposed solutions with those of human programmers on a set of programming tasks. For the former, we assess the performance and functionality of Copilot in solving selected fundamental problems in computer science, like sorting and implementing data structures. In the latter, a dataset of programming problems with human-provided solutions is used. The results show that Copilot is capable of providing solutions for almost all fundamental algorithmic problems, however, some solutions are buggy and non-reproducible. Moreover, Copilot has some difficulties in combining multiple methods to generate a solution. Comparing Copilot to humans, our results show that the correct ratio of humans' solutions is greater than Copilot's suggestions, while the buggy solutions generated by Copilot require less effort to be repaired. Based on our findings, if Copilot is used by expert developers in software projects, it can become an asset since its suggestions could be comparable to humans' contributions in terms of quality. However, Copilot can become a liability if it is used by novice developers who may fail to filter its buggy or non-optimal solutions due to a lack of expertise.

Keywords: Code Completion, Language Model, GitHub Copilot, Testing.

1. Introduction

Recent breakthroughs in Deep Learning (DL), in particular the Transformer architecture, have revived the Software Engineering (SE) decades-long dream of automating code generation that can speed up programming activities. Program generation aims to deliver a program that meets a user's intentions in the form of input-output examples, natural language descriptions, or partial programs [2, 33, 25].

Program synthesis is useful for different purposes such as teaching, programmer assistance, or the discovery of new algorithmic solutions for a problem [25]. One finds different approaches to automatic code generation in the literature, from natural language programming [35] and formal models [15, 27] to Evolutionary Algorithms [48] and machine-learned translation [42].

Novel Large Language Models (LLMs) with the transformer architecture recently achieved good performance in automatic program synthesis [6, 8, 9, 20]. One such model is Codex [8]; a GPT-3 [6] based language model with up to 12 billion parameters which has been pretrained on 159 GB of code samples from 54 million GitHub repositories. Codex shows a good performance in solving a set of hand-written programming problems (i.e., not in the training dataset) using Python, named HumanEval dataset [8]. This dataset includes simple programming problems with test cases to assess the functional correctness of codes. A production version of Codex is available as an extension on the Visual Studio Code development environment, named GitHub Copilot[1]. Copilot, as an "AI pair programmer", can generate code in different programming languages when provided with some context (called prompt), such as comments, methods names, or surrounding code.

Several studies focus on the correctness of codes sug-

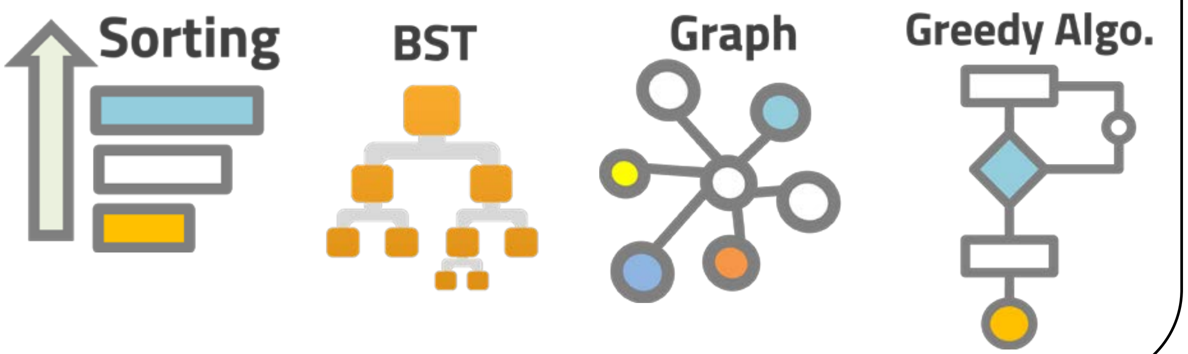
*Corresponding authors. Both authors contributed equally to this research.

Email addresses: {arghavan.moradi-dakhel, vahid.majdinasab, amin.nikanjam, foutse.khomh, michel.desmarais}@polymtl.ca (Amin Nikanjam, Foutse Khomh, Michel C. Desmarais), zmjiang@cse.yorku.ca (Zhen Ming (Jack) Jiang)

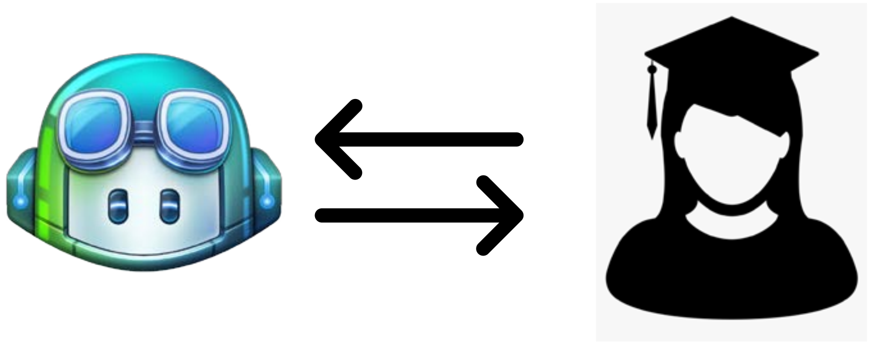
[1] https://copilot.github.com/

Preprint submitted to Journal of Software and System — *April 18, 2023*

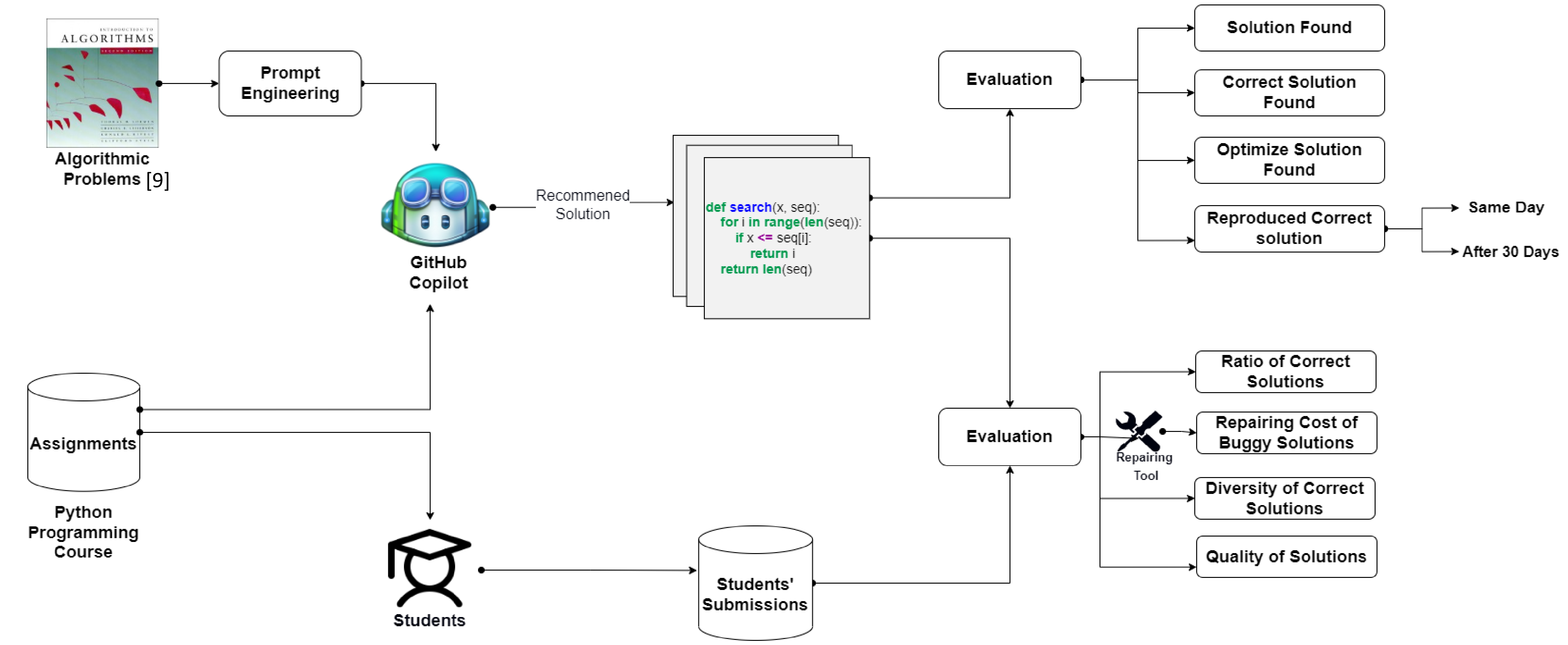
RQ1: Correctness, Reproducibility and Optimality on fundamental algorithmic problems



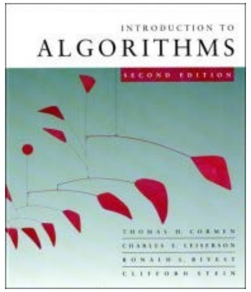
RQ2: Competitive with human solutions in different aspects



Methodology



RQ1: Fundamental Algorithmic Problems

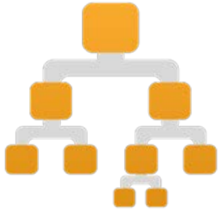


20 Tasks

Sorting

8 different sorting algorithm from easy to hard

BST

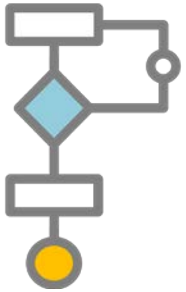


Find min an max, walks and finding successor node

Graph

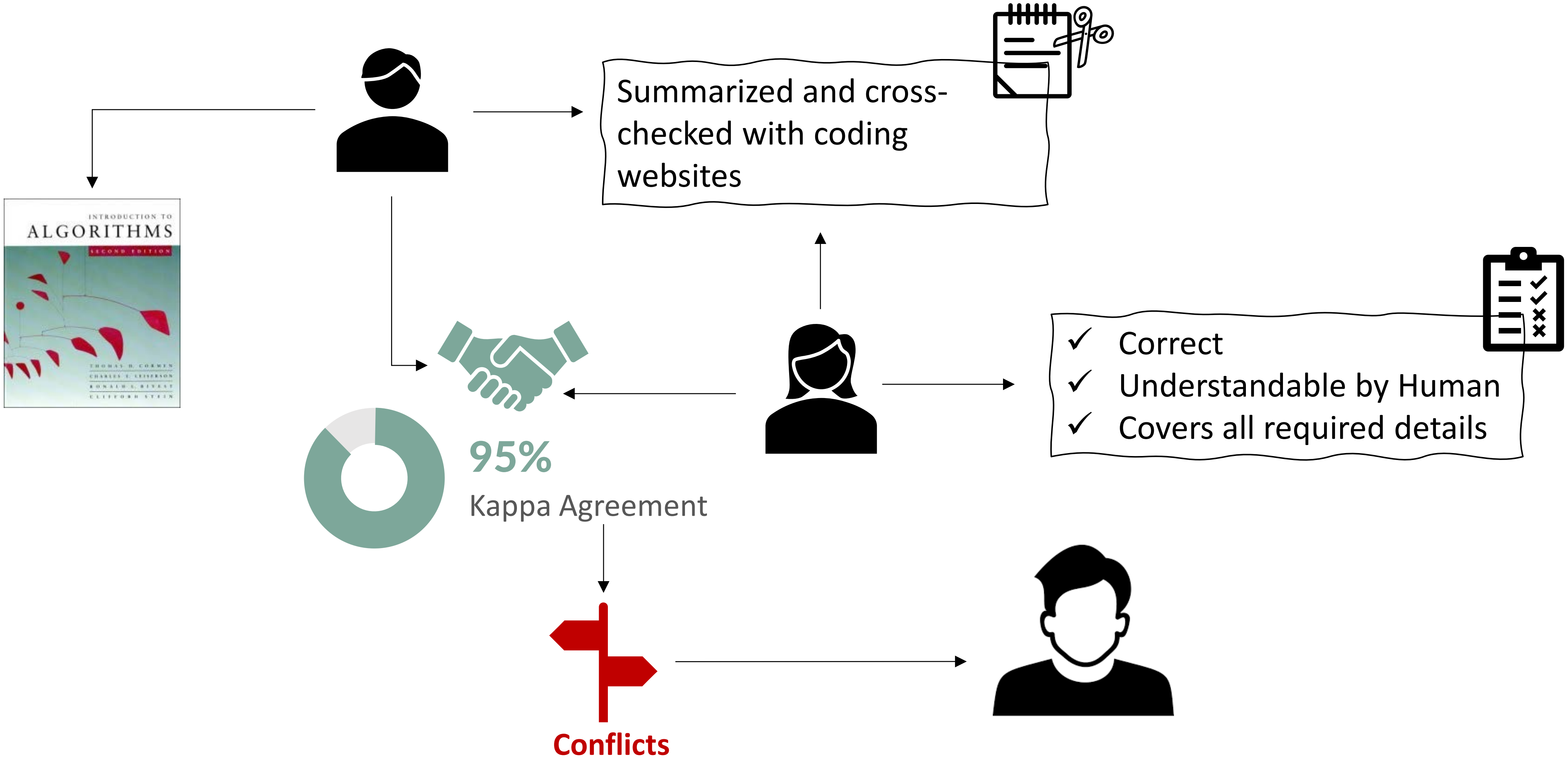
Simple Graph, DAG, BFS and DFS

Greedy Algo.

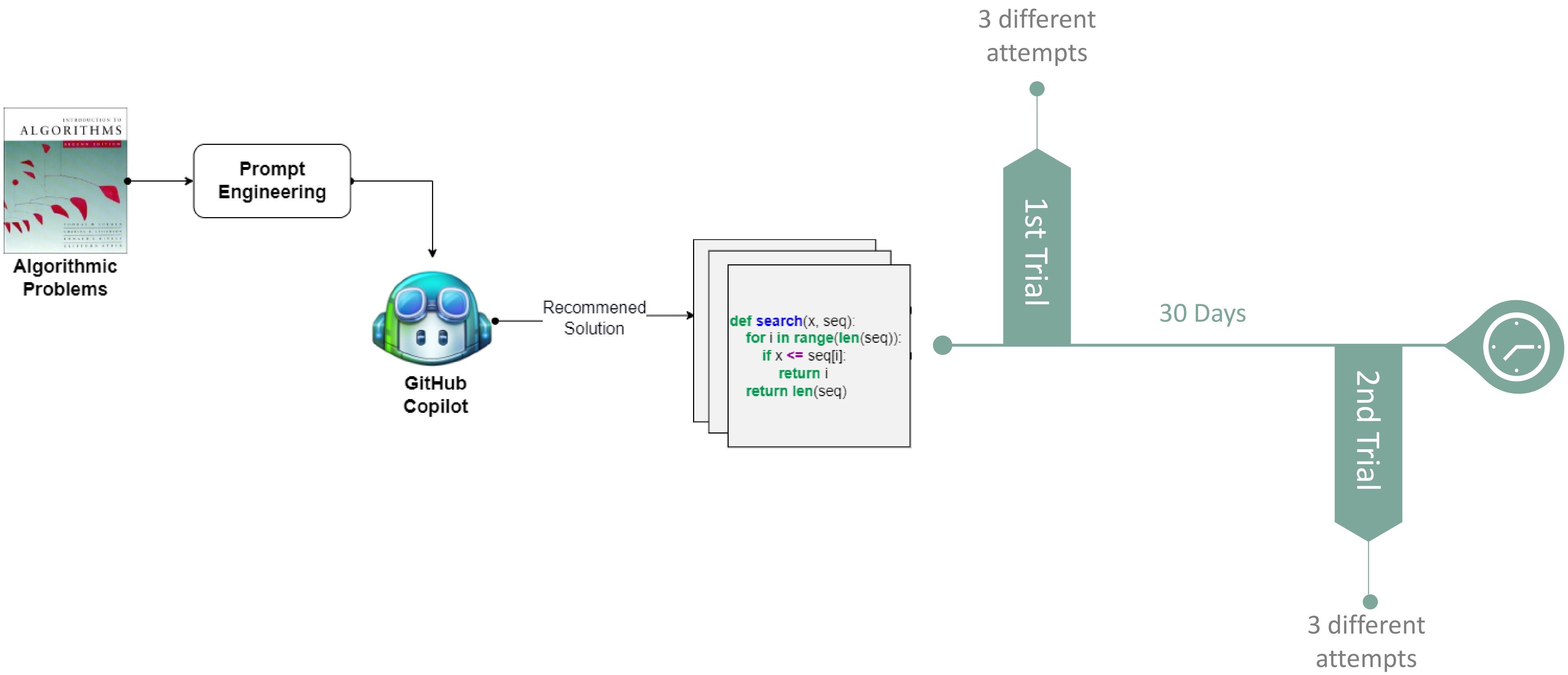


Activity selection class

RQ1: Prompt Engineering



RQ1: Generate Solutions

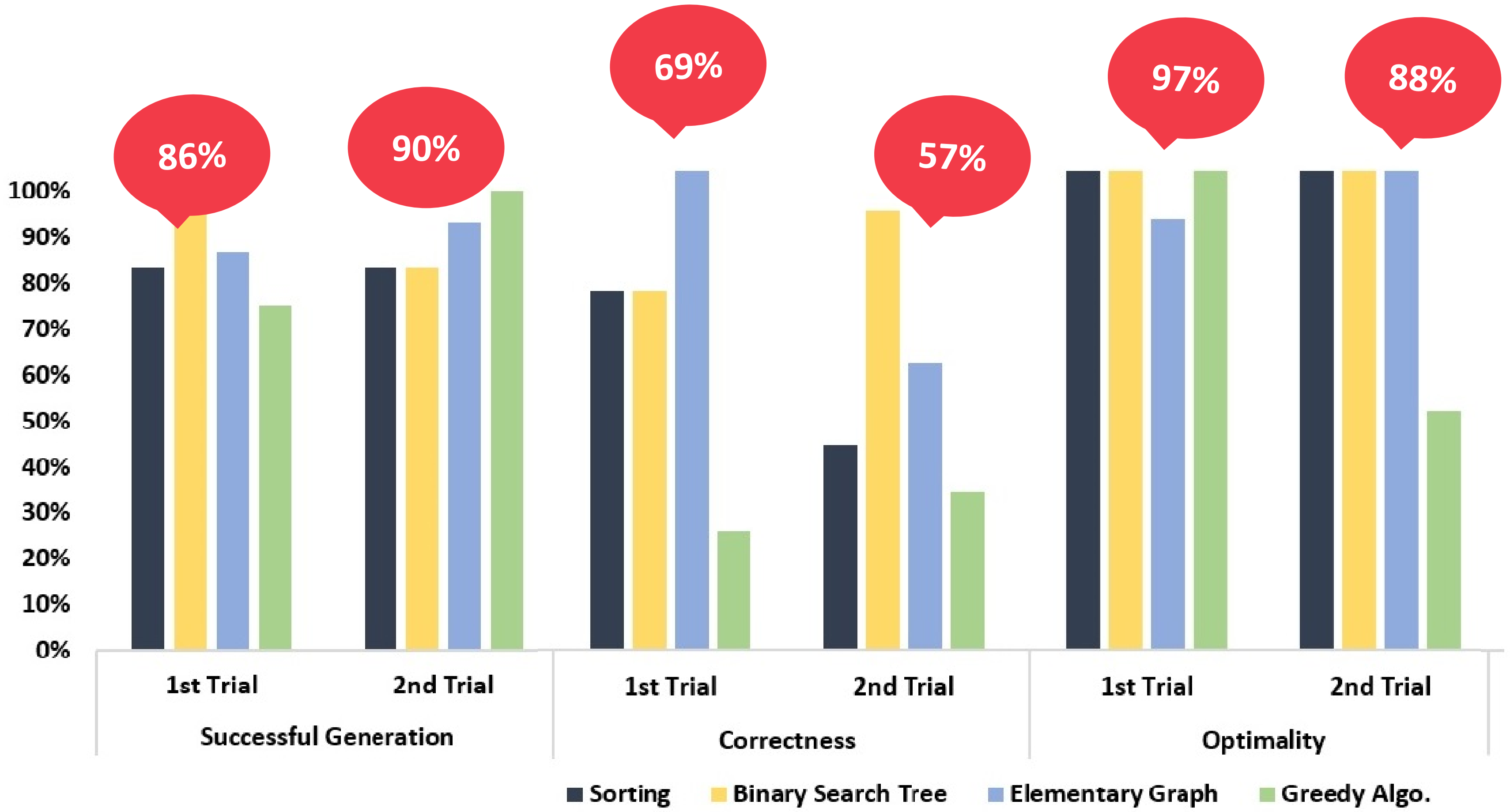


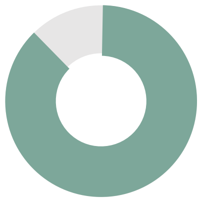
RQ1: Evaluation-I

Response Received

Functional Correctness

Optimality



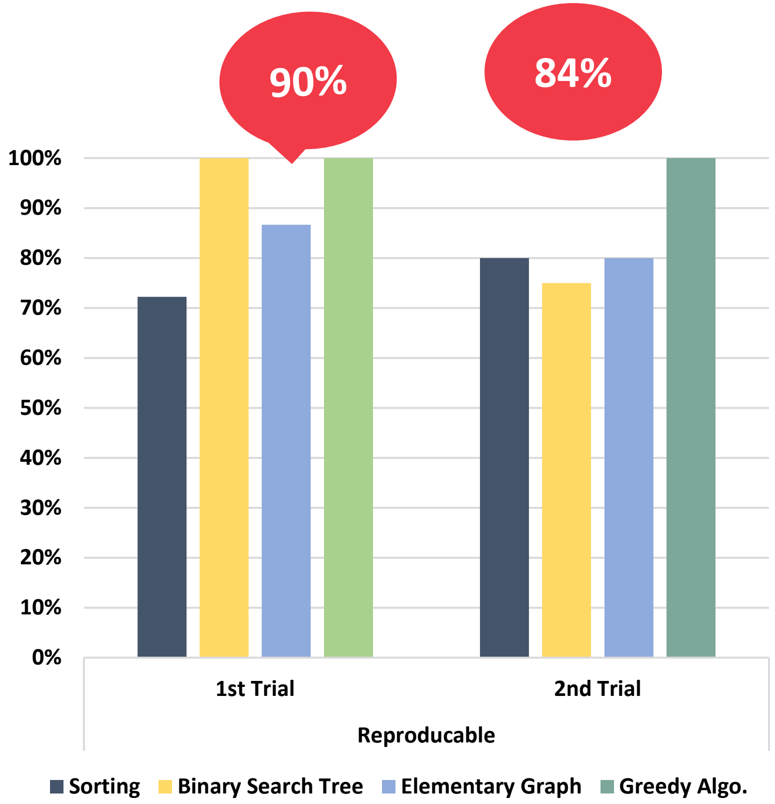


93%

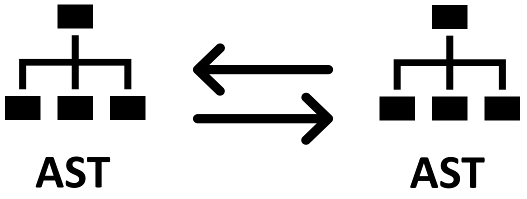
Kappa Agreement

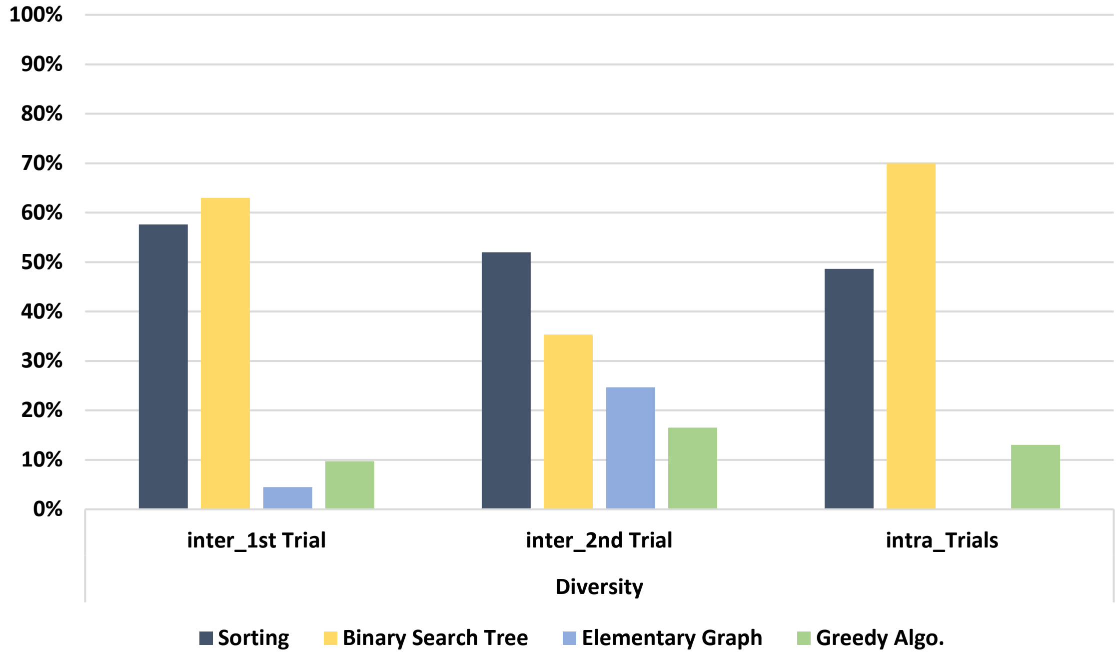
RQ1: Evaluation-II

Reproducibility of Correct Solutions

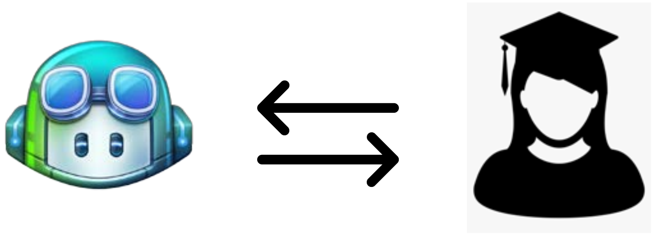


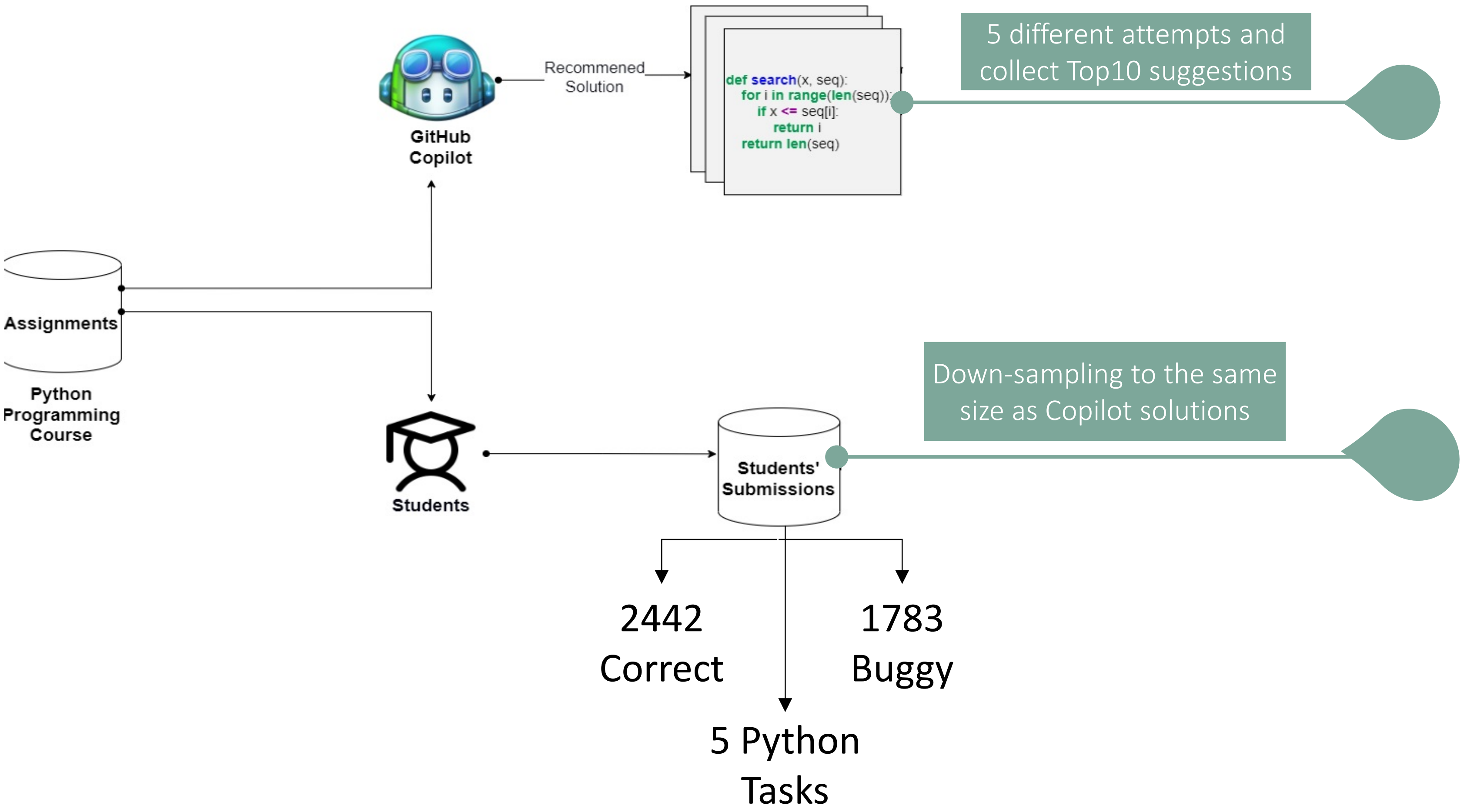
Diversity of Solutions (inter/intra Trial(s))





RQ2: Compare Copilot and Human



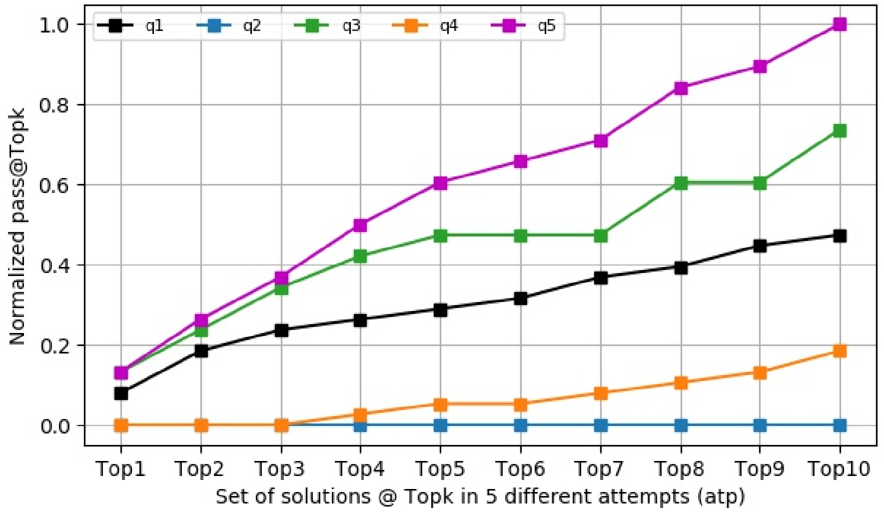


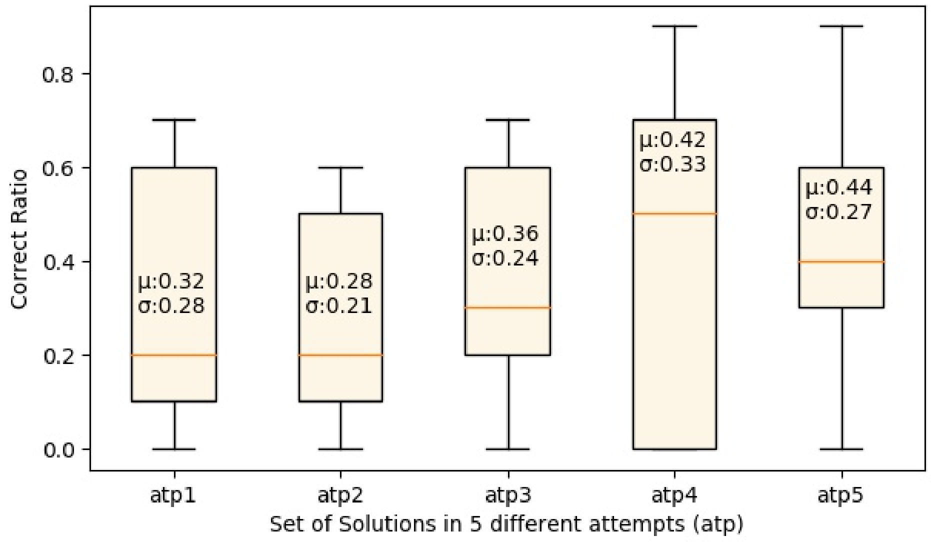
RQ2: Evaluation-Correct Ratio (pass@Topk)

…"solve the problem by implementing 3 different functions"…

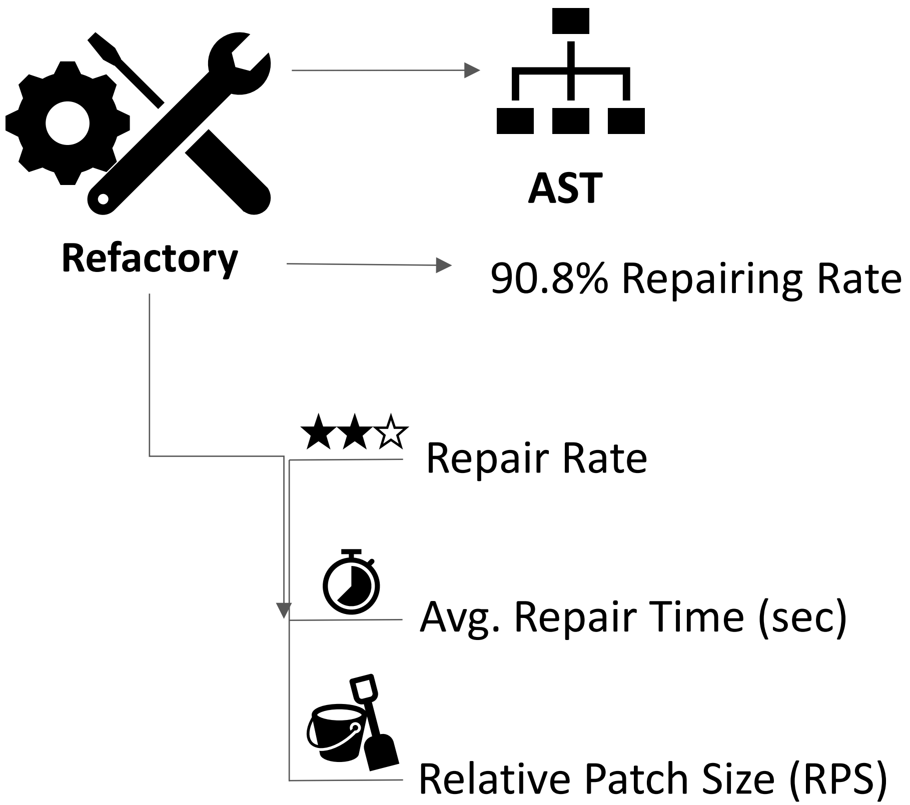
…"put the older people at top of the list"…

		Copilot			Students
	Task	CR@Top1	CR@Top5	CR@Top10	CR
q1	Sequential Search	**0.6**	0.44	0.36	0.57
q2	Unique Dates Months	0.00	0.00	0.00	**0.40**
q3	Duplicate Elimination	**1**	0.72	0.56	0.64
q4	Sorting Tuples	0.00	0.08	0.14	**0.54**
q5	Top-k Elements	**1**	0.92	0.76	0.79
	Total	0.52	0.43	0.35	**0.59**

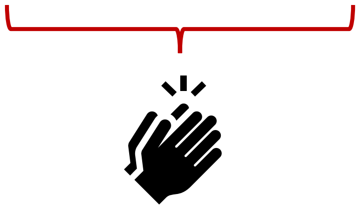




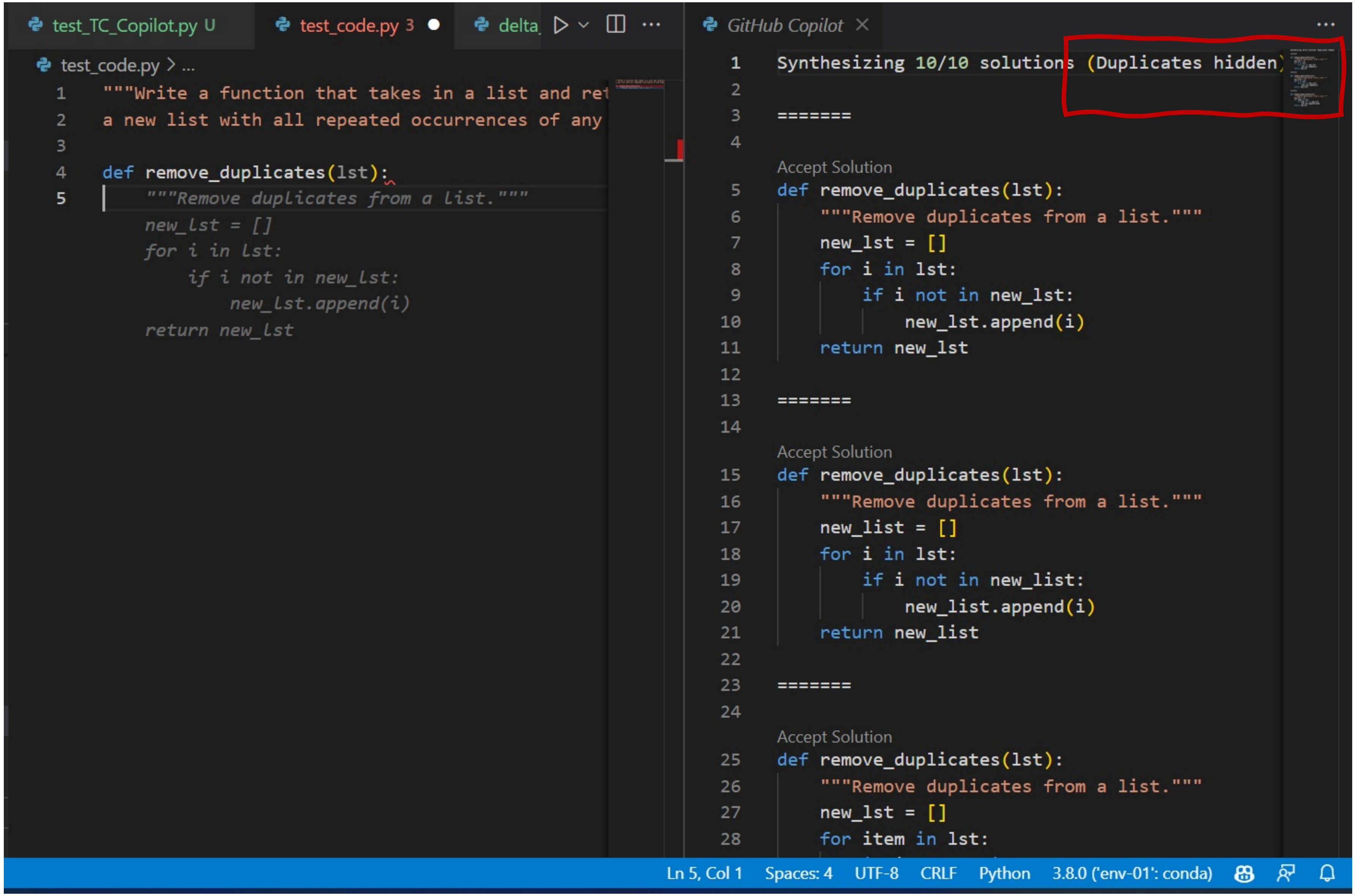
RQ2: Evaluation-Repairing Cost of Buggy Solutions



Task		Copilot			Students		
		Rep Rate	Avg Rep Time(sec)	Avg rps	Rep Rate	Avg Rep Time	Avg RPS
q1	sequential search	0.94	9.61	0.48	0.98	2.58	0.40
q2	unique dates months	0.92	3.26	0.28	0.82	3.81	0.44
q3	duplicate elimination	0.91	0.64	0.26	0.96	4.35	0.30
q4	sorting tuples	**1.00**	**0.78**	**0.15**	0.85	8.82	0.29
q5	top-k elements	**1.00**	10.40	0.50	0.85	12.84	0.30
Total		**0.95**	**4.94**	**0.33**	0.89	6.48	0.35

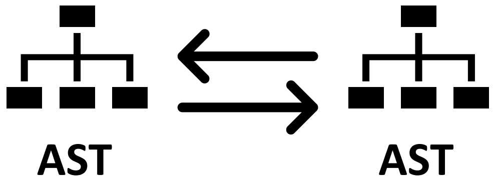


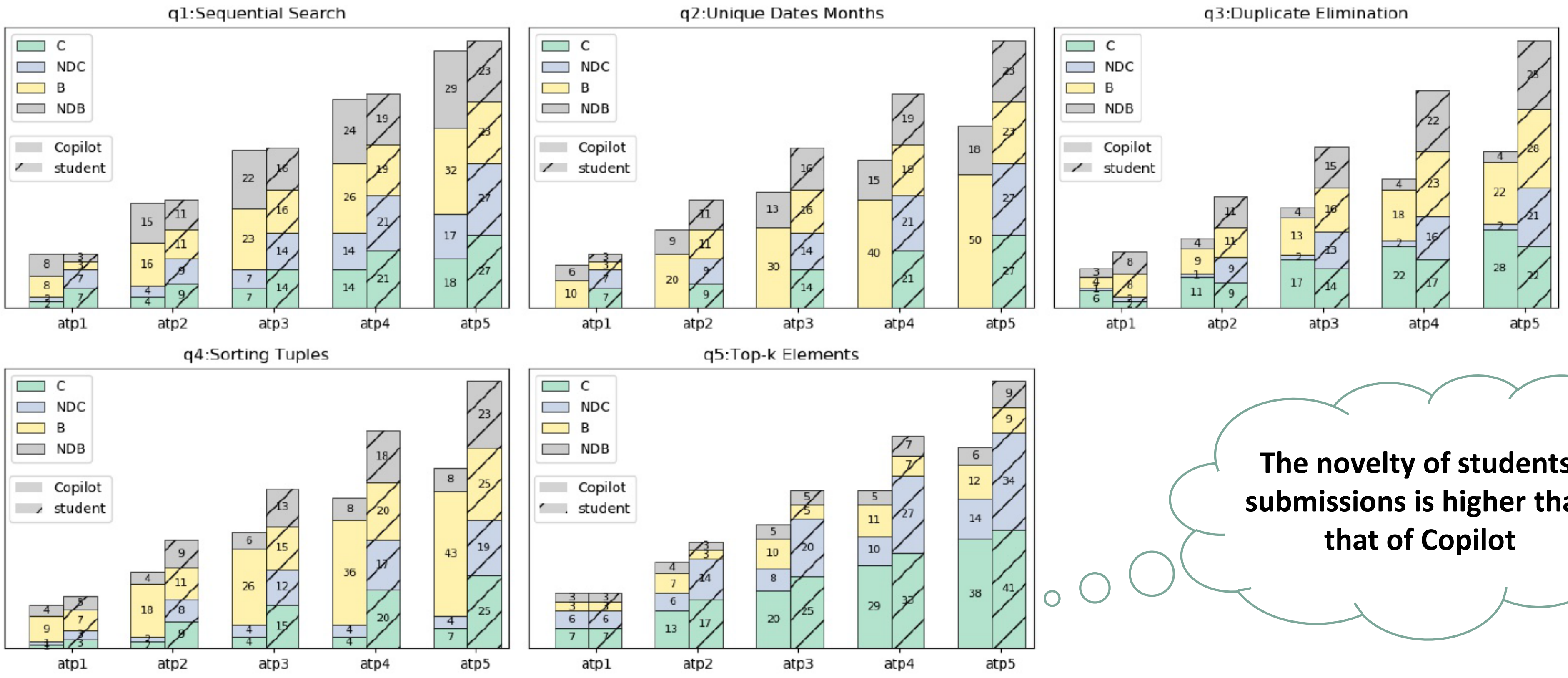
RQ2: Evaluation-Diversity of Solutions-I



RQ2: Evaluation-Diversity of Solutions-II

The novelty of Copilot in solving the same problem compared to students





The novelty of students' submissions is higher than that of Copilot

The cumulative distribution of solutions by Copilot and students. It shows the cumulative distribution of Correct (C), None Duplicate Correct (NDC), Buggy (B) and None Duplicate Buggy (NDB) solutions for Copilot and students. Attempts (atp) for students equals to the sampleset of randomly selection of their submission. The growth of NDC solutions for Copilot's solutions decreases or stops for some programming tasks while the number of its Correct (C) solutions increases. The diversity of submissions for students is greater than Copilot's solutions.

RQ2: Evaluation-Cyclomatic Complexity (C.C)

Question	C.C. Copilot	C.C. Students
Sequential Search	5.8 ± 1.94	4.63 ± 2.1
unique dates Months	-	4.18 ± 1.03
Duplicate Elimination	3 ± 0.01	3.12 ± 0.5
Sorting Tuples	1 ± 0	4.13 ± 1.03
Top_k Elements	1.44 ± 0.69	3.3 ± 1.46
Total	2.81	3.87

Don't use "sort" or "sorted"

In summary

Despite low correct ratio and diversity, generates optimal solutions

Reproducible

Struggling in understanding natural language utterances i.e., putting limit for variable

“...older people are at the front...”

“...descending order...”

CR: 14% -> 79%

Little changes to convert a buggy code into the correct one

95% Repairing Rate

Despite their performance, they are not yet trustworthy!

We observed model's mistakes that never occurred in human codes

```python
if not logfile.exists():
        logging.basicConfig(
                filename=str(logf),
                format='%(asctime)s %(levelname)s %(name)s %(message)s',
                datefmt='%Y-%m-%d %H:%M:%S',
                level=logging.DEBUG,
        )
else:
        logging.basicConfig(
                filename=str(logf),
                format='%(asctime)s %(levelname)s %(name)s %(message)s',
                datefmt='%Y-%m-%d %H:%M:%S',
                level=logging.DEBUG,
        )
```

Repeat the same statement for both if and else.

Adding imports AFTER the function to create. Those imports have no link to the function to implement

```python
import sys

from PyQt5.QtCore import QObject, QThread, pyqtSignal, pyqtSlot, QUrl
from PyQt5.QtWidgets import QApplication, QMainWindow, QActionGroup

import resources.icons as icons
from resources.logger import logger
```

We observed model's mistakes that never occurred in human codes

```python
def vertex3tuple(vertices):
    """return 3 points for each vertex of the polygon. This will include the vertex and the 2 points on both sides of the

      polygon with vertices ABCD
      Will return
      DAB, ABC, BCD, CDA -> returns 3tuples
      #A    B    C    D  -> of vertices
    """

      return [(vertices[-1], vertices[0], vertices[1]),
                   (vertices[0], vertices[1], vertices[2]),
                   (vertices[1], vertices[2], vertices[3]),
                   (vertices[2], vertices[3], vertices[0])]
```

This will only work if they are 4 vertices (just as in the provided docstring, i.e. prompt). It doesn't work in other cases.

- ✓Given the increasing adoption of LLMs.
- ✓Given that the effectiveness of popular quality assurance techniques like mutation testing depends on a precise characterization of faults occurring in the code under test.

Would existing QA techniques cope efficiently with LLM generated code?

We believe that there is a need for a precise characterization of faults contained in LLM-generated code!

Noname manuscript No.
(will be inserted by the editor)

Bugs in Large Language Models Generated Code: An Empirical Study

Florian Tambon* · Arghavan Moradi Dakhel* · Amin Nikanjam · Foutse Khomh · Michel C. Desmarais · Giuliano Antoniol

Codex

HUAWEI PanGu-Coder

CODEGEN

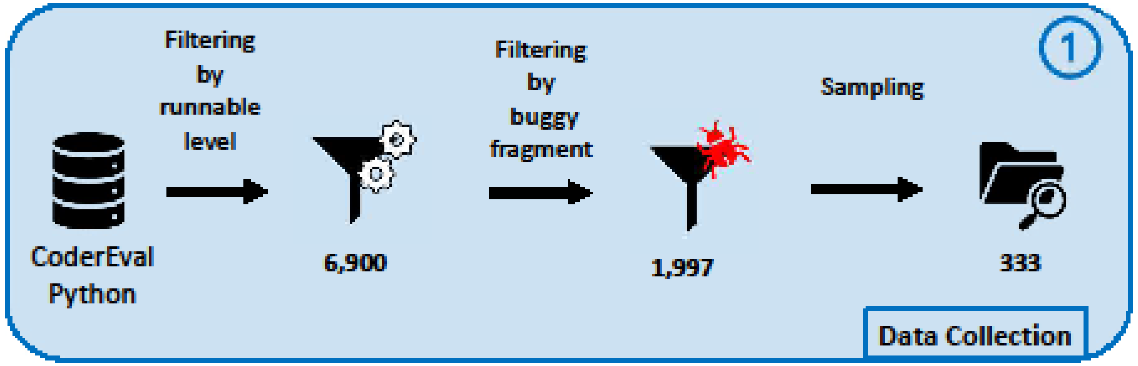
Coder Eval

230 functions from 43 Python projects and 230 methods from 10 Java projects.

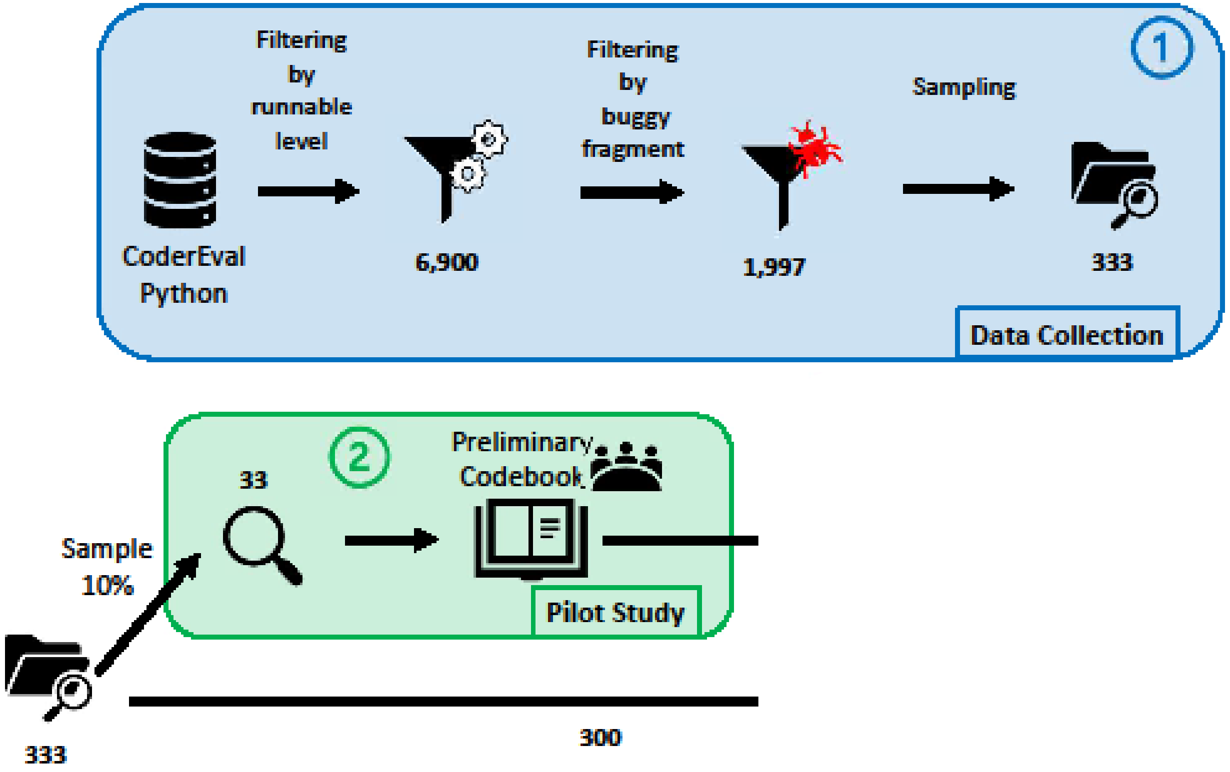
RQ1: What are the characteristics of bugs occurring in code generated by LLMs for real-world project tasks?

RQ2: To what extent are the identified bug patterns in LLM-generated code relevant for software practitioners working with LLMs?

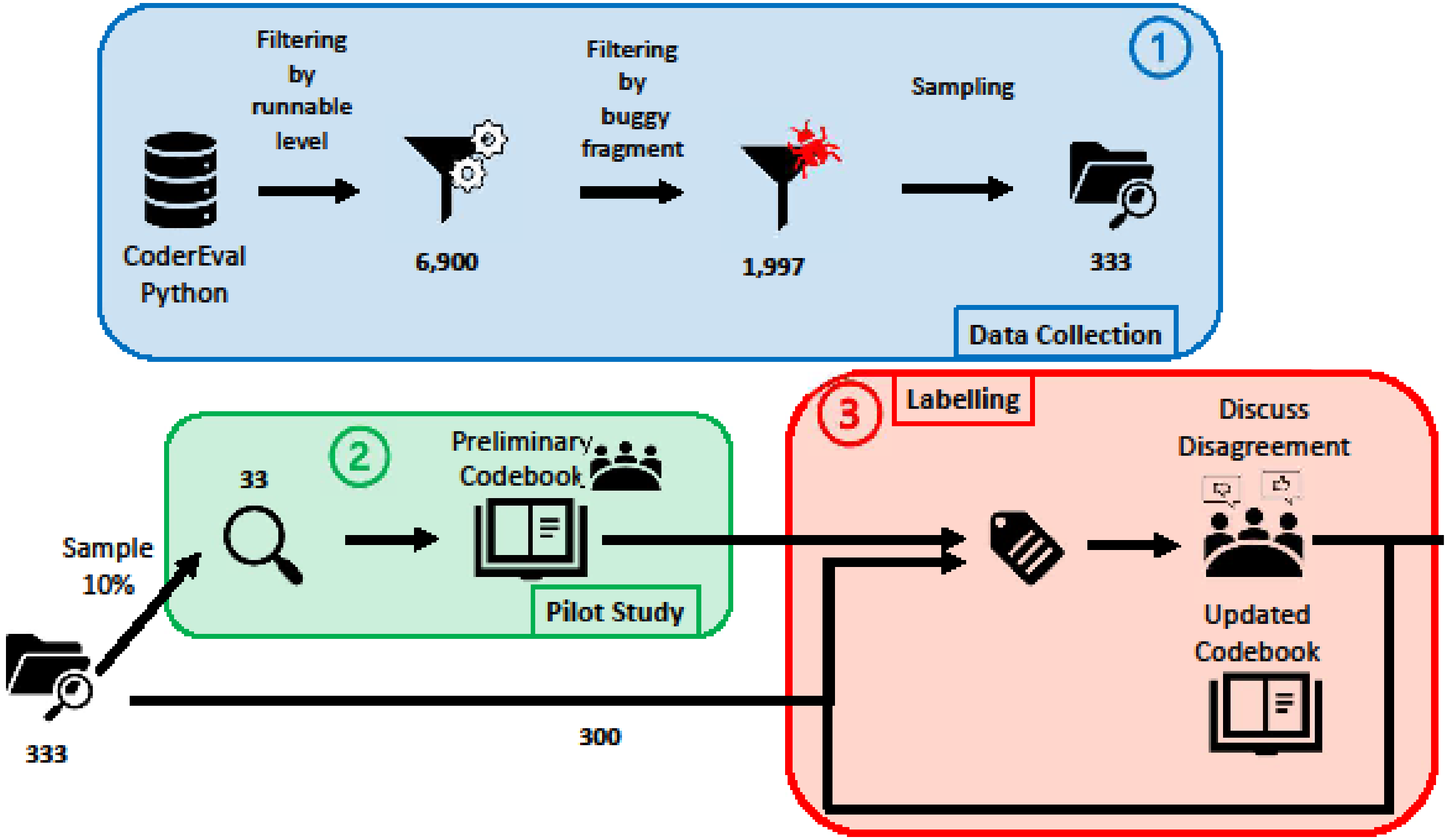
Methodology



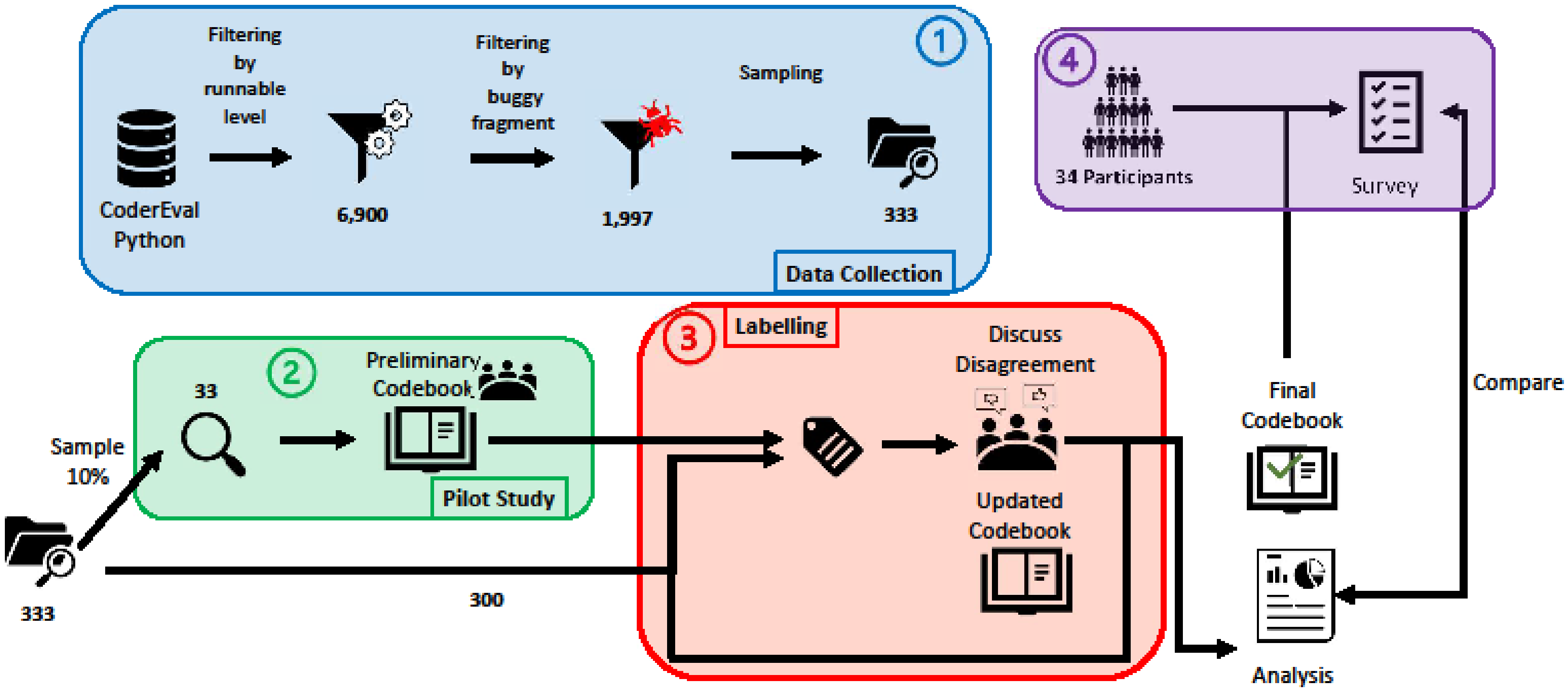
Methodology



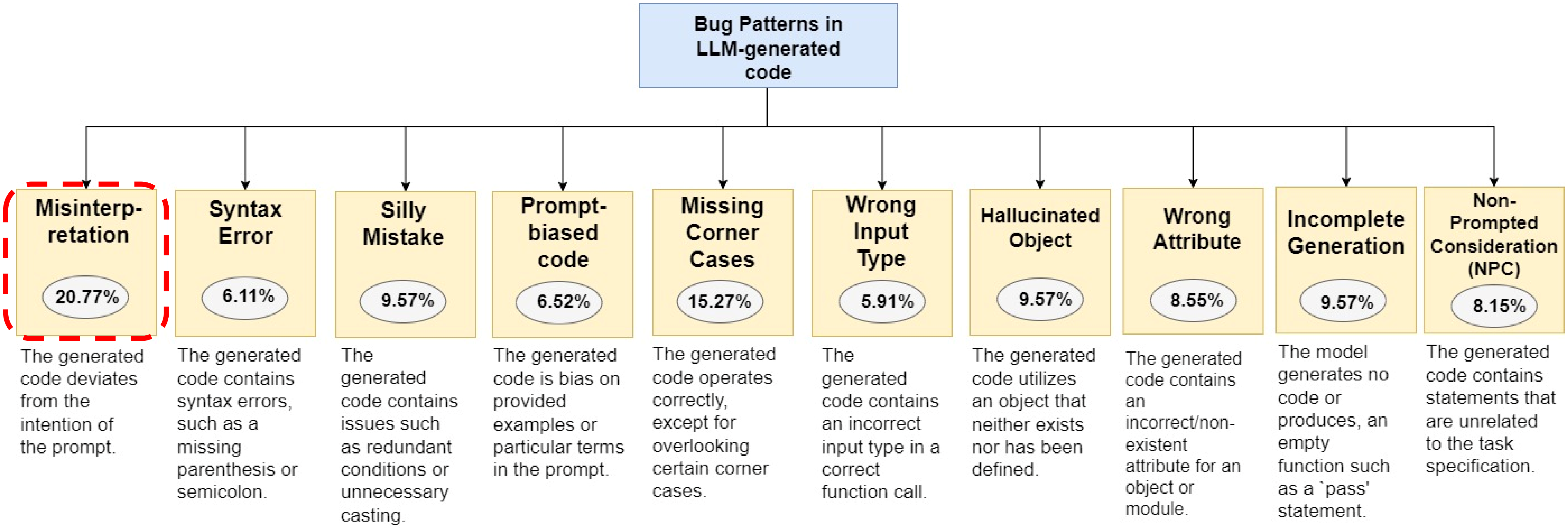
Methodology

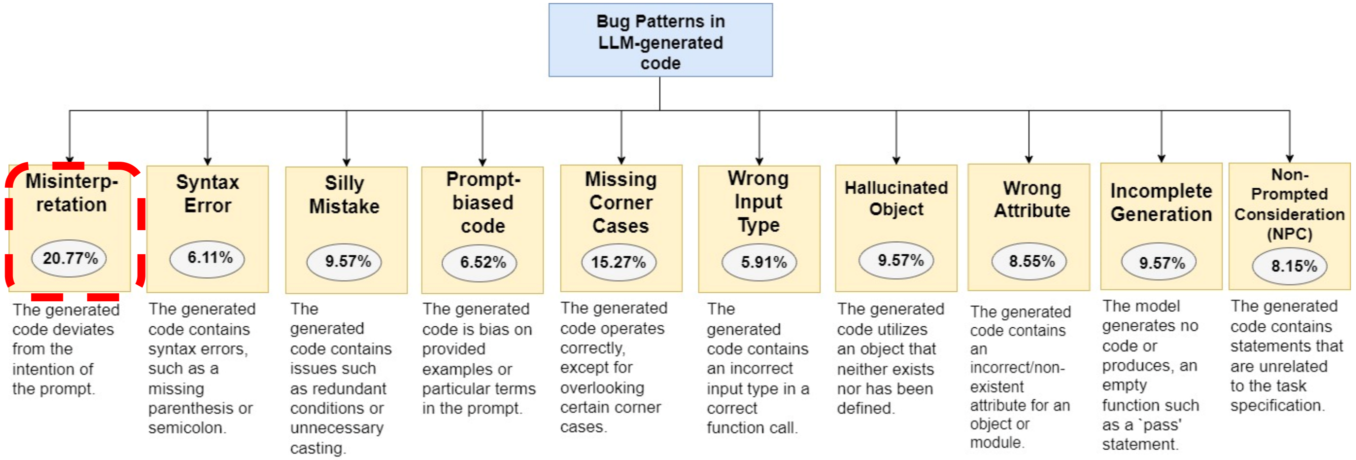


Methodology



The Taxonomy



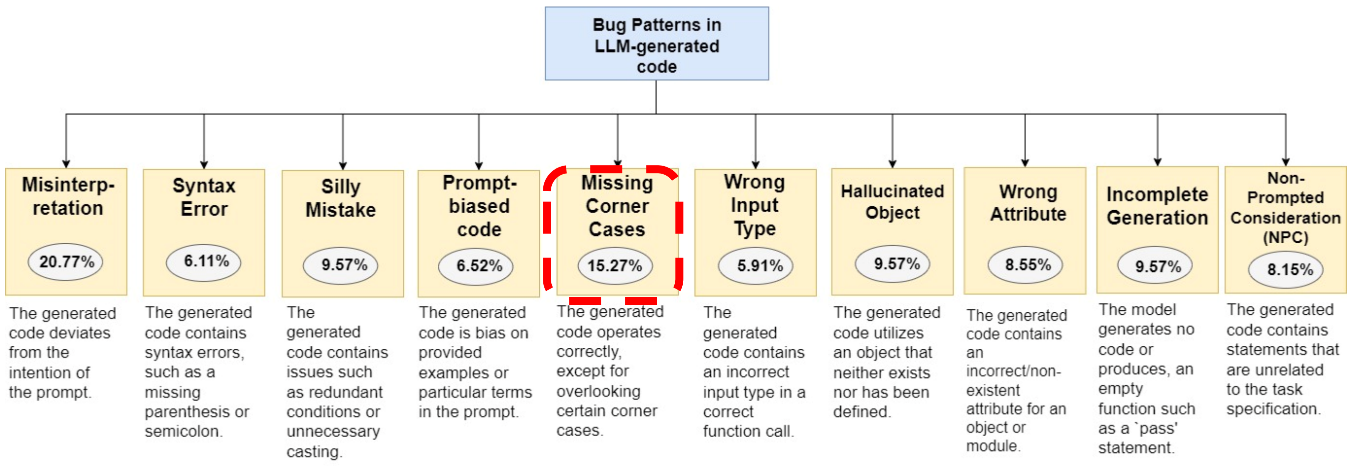



```
def int_to_string(number: int, alphabet: List[str], padding: Optional[int] = None) -> str:
    """
    Convert a number to a string, using the given alphabet.

    The output has the most significant digit first.
    """
    output = ""
    alpha_len = len(alphabet)
    while number:
        number, digit = divmod(number, alpha_len)
        output += alphabet[digit]
    if padding:
        remainder = max(padding - len(output), 0)
        output = output + alphabet[0] * remainder
    return output[::-1]
```

Reference solution

```
def int_to_string(number: int, alphabet: str) -> str:
    return alphabet[number]
```

Solution proposed By Pan PanGu-Coder

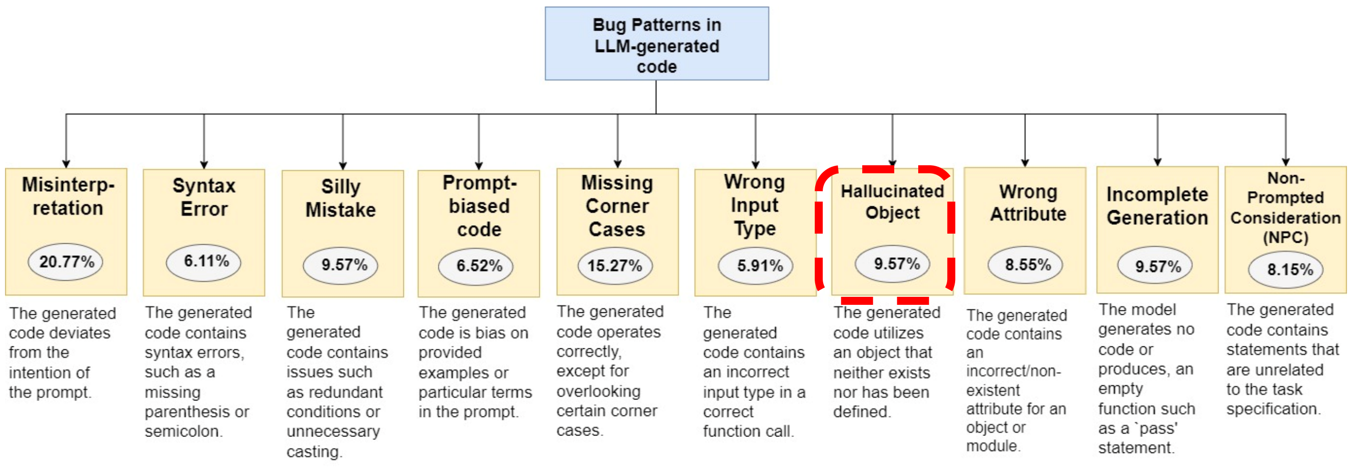


Reference solution

```
import socket
def is_local(host):
    """
    Checks if the host is the localhost

    :param host: The hostname or ip
    :return: True if the host is the localhost
    """
    return host in ["127.0.0.1",
                    "localhost",
                    socket.gethostname(),
                    # just in case socket.gethostname() does not work
we also try the following:
                    platform.node(),
                    socket.gethostbyaddr(socket.gethostname())[0]
                    ]
```

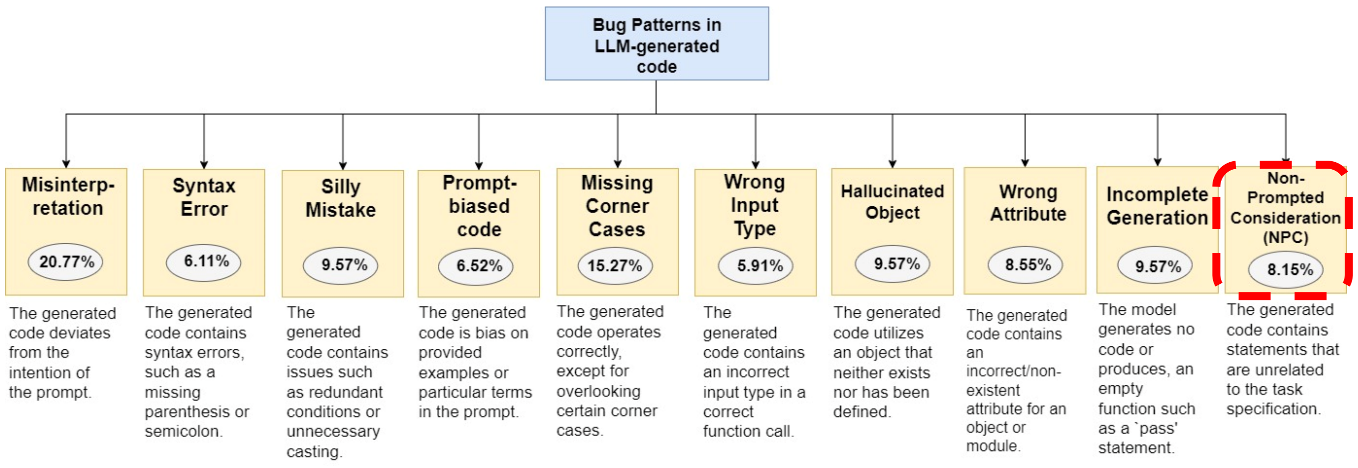
Solution proposed By Codex missed Some relevant options

```
import socket
def is_local(host):
    """
    Checks if the host is the localhost

    :param host: The hostname or ip
    :return: True if the host is the localhost
    """
    return host in ['localhost', '127.0.0.1', '::1']
```




```
def make_find_paths(find_paths):
    '''
    Given a sequence of path fragments or patterns as passed to `--find`, transform all
    path fragments into glob patterns. Pass through existing patterns untouched.

    For example, given find_paths of:

      ['foo.txt', 'pp:root/somedir']

    ... transform that into:

      ['sh:**/*foo.txt*/**', 'pp:root/somedir']
    '''
    return [find_path_to_glob(x) for x in find_paths]
```

Solution proposed By Codex calls an undefined function "find_path_to_ glob"

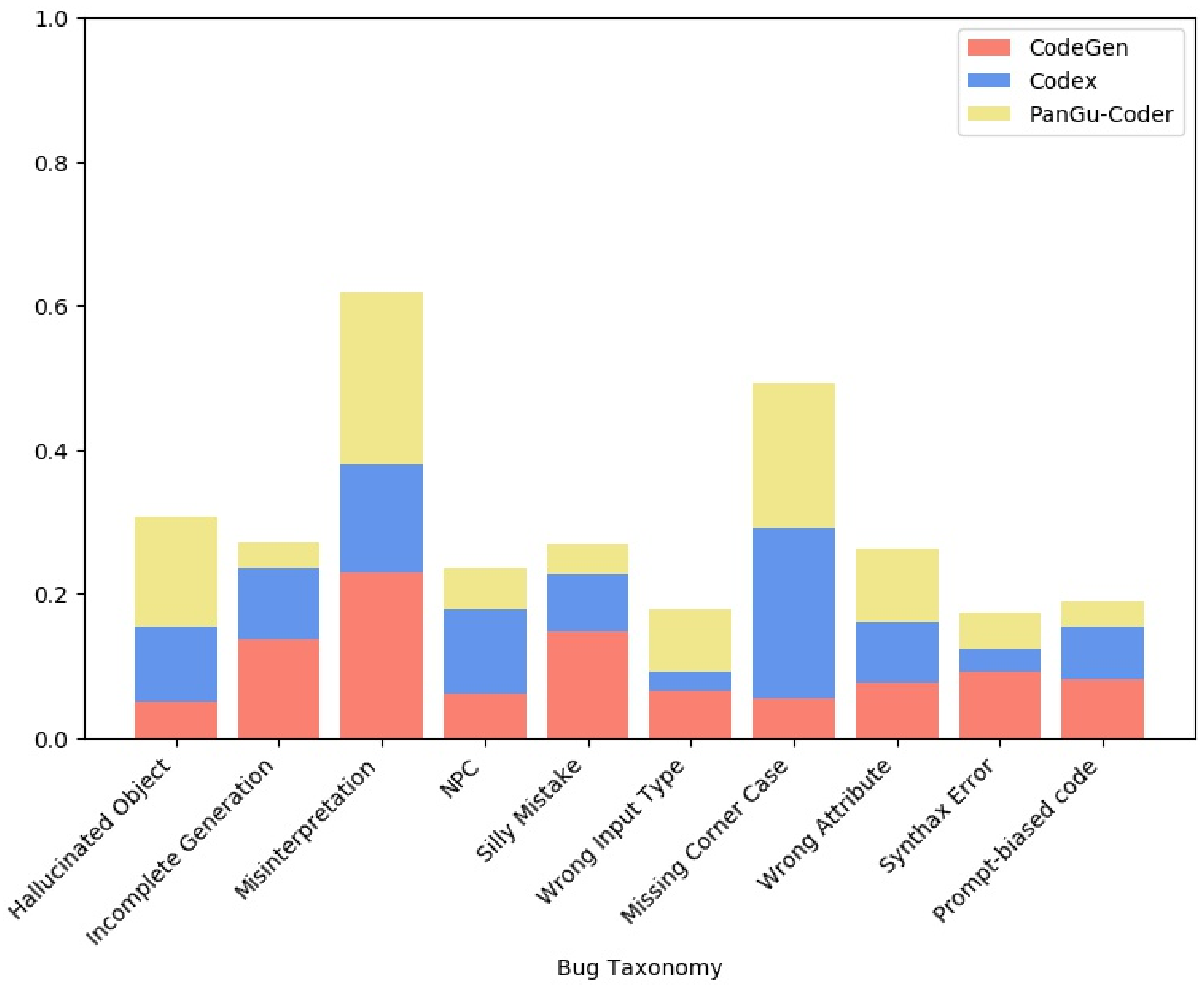



```python
def parser_flags(parser):
    '''
    Given an argparse.ArgumentParser instance, return its argument flags in a space—separated
    string.
    '''
    flags = []
    for action in parser._actions:
        if isinstance(action, argparse._SubParsersAction):
            for choice, subparser in action.choices.items():
                flags.append(choice)
                flags += parser_flags(subparser)
        else:
            flags.append(action.option_strings[0])
    return " ".join(sorted(flags))
```

Solution proposed By Codex sorts the list of flags before joining them, which is not in line with the given prompt

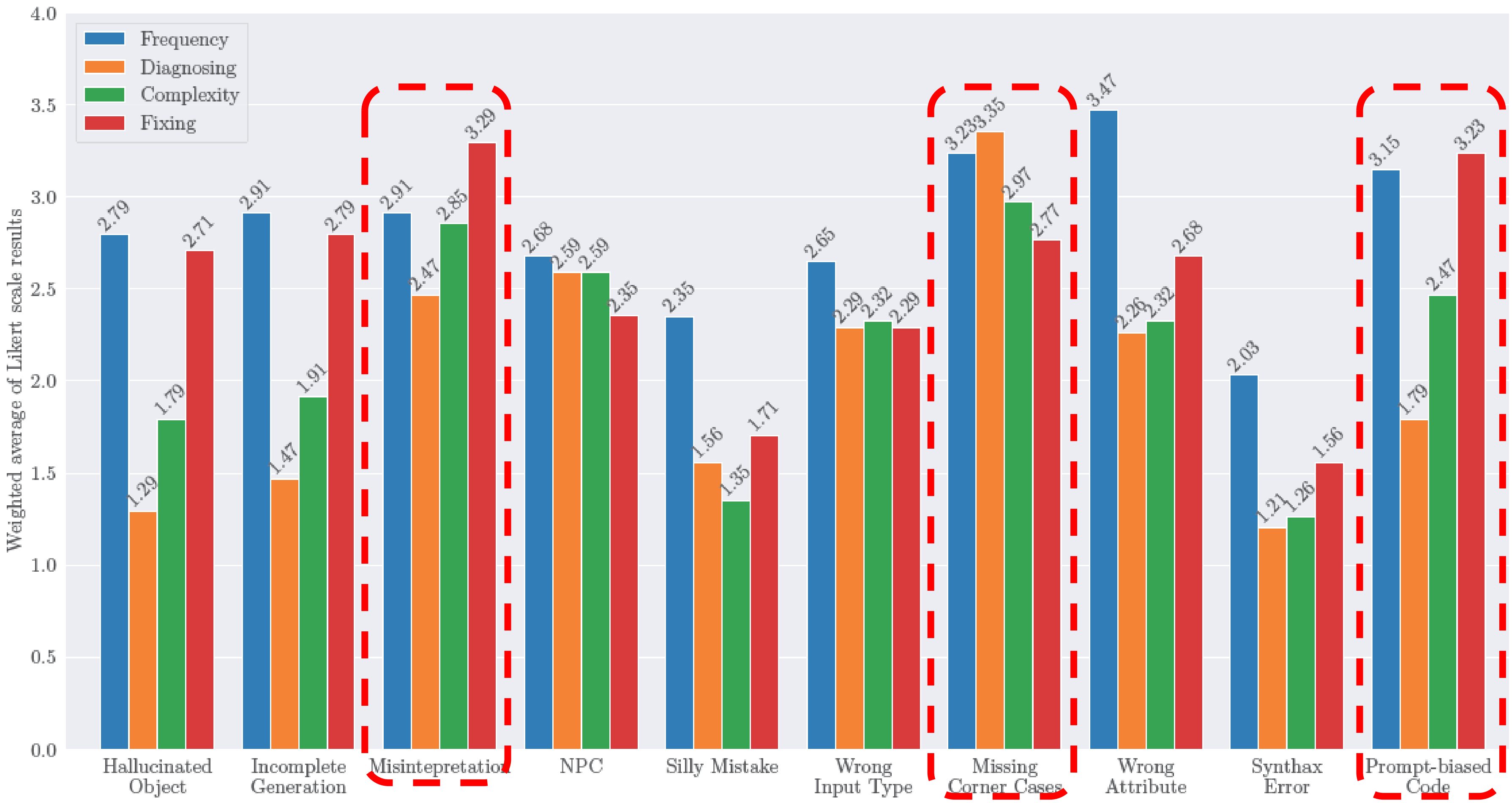
The Taxonomy

Distribution of Bug types in LLMs

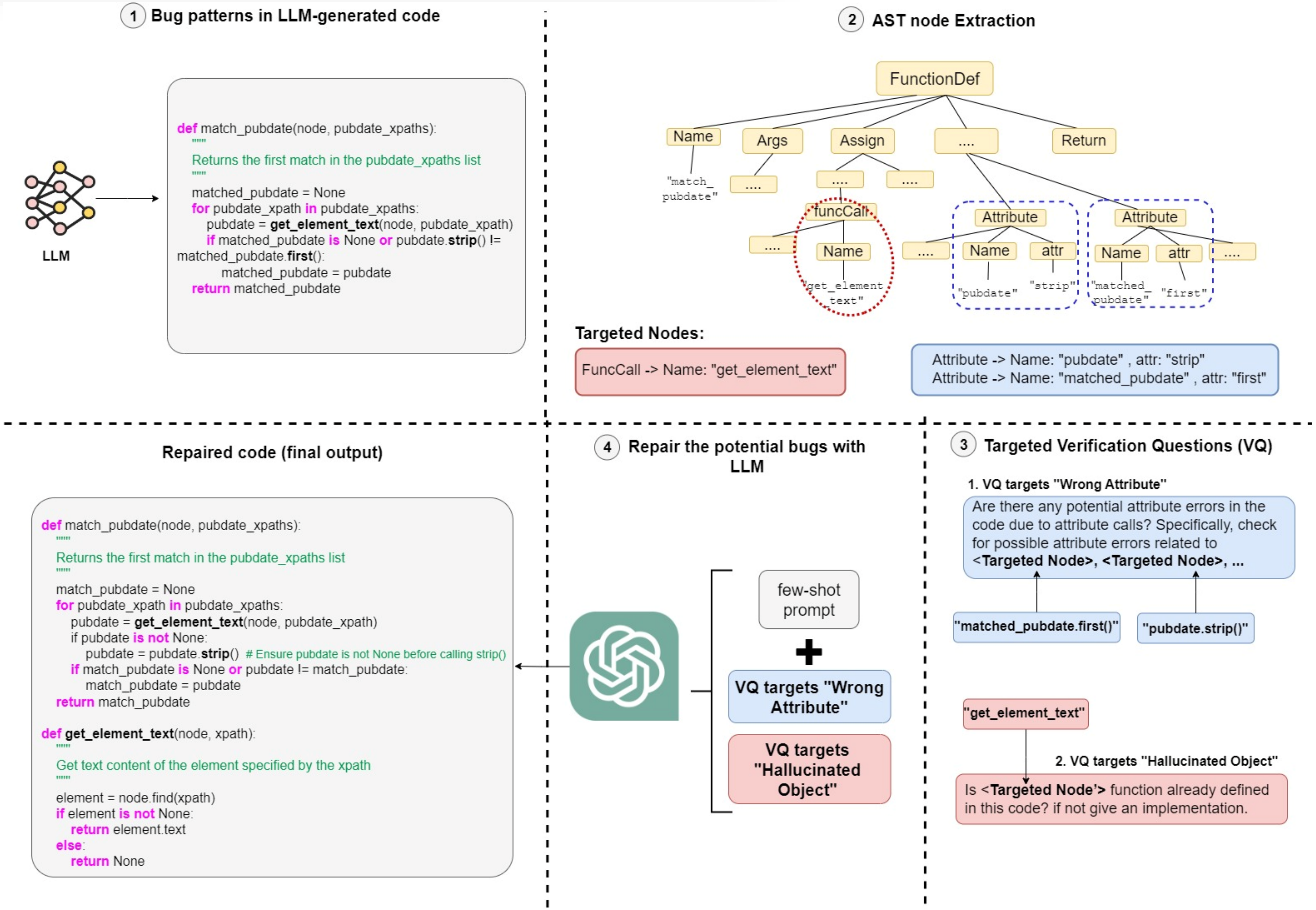


The Taxonomy

Survey participants assessment of the bug patterns



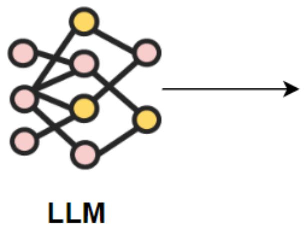
Chain of Targeted Verification Questions to Improve the Reliability of Code Generated by LLMs



Generation of Verification Questions

1 **Bug patterns in LLM-generated code**

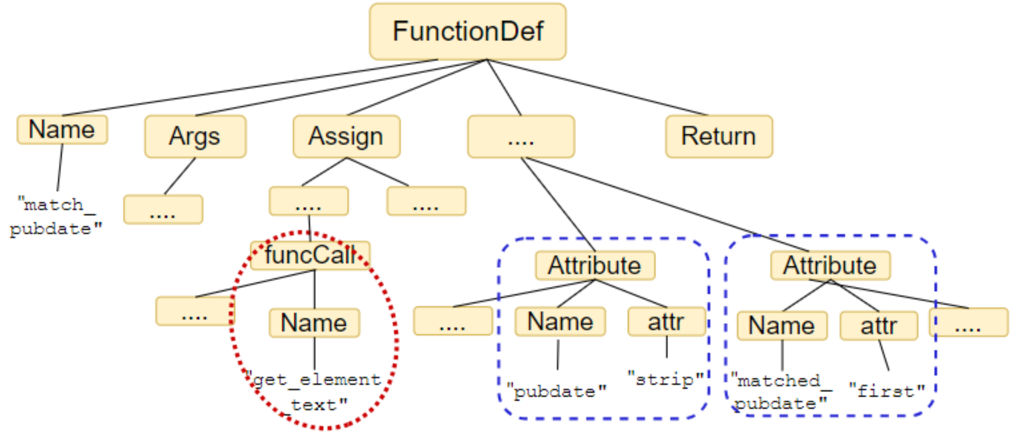
is an "Hallucinated Object"




```
def match_pubdate(node, pubdate_xpaths):
    """
    Returns the first match in the pubdate_xpaths list
    """
    matched_pubdate = None
    for pubdate_xpath in pubdate_xpaths:
        pubdate = get_element_text(node, pubdate_xpath)
        if matched_pubdate is None or pubdate.strip() !=
matched_pubdate.first():
            matched_pubdate = pubdate
    return matched_pubdate
```

is a "Wrong Attribute"

2 **AST node Extraction**



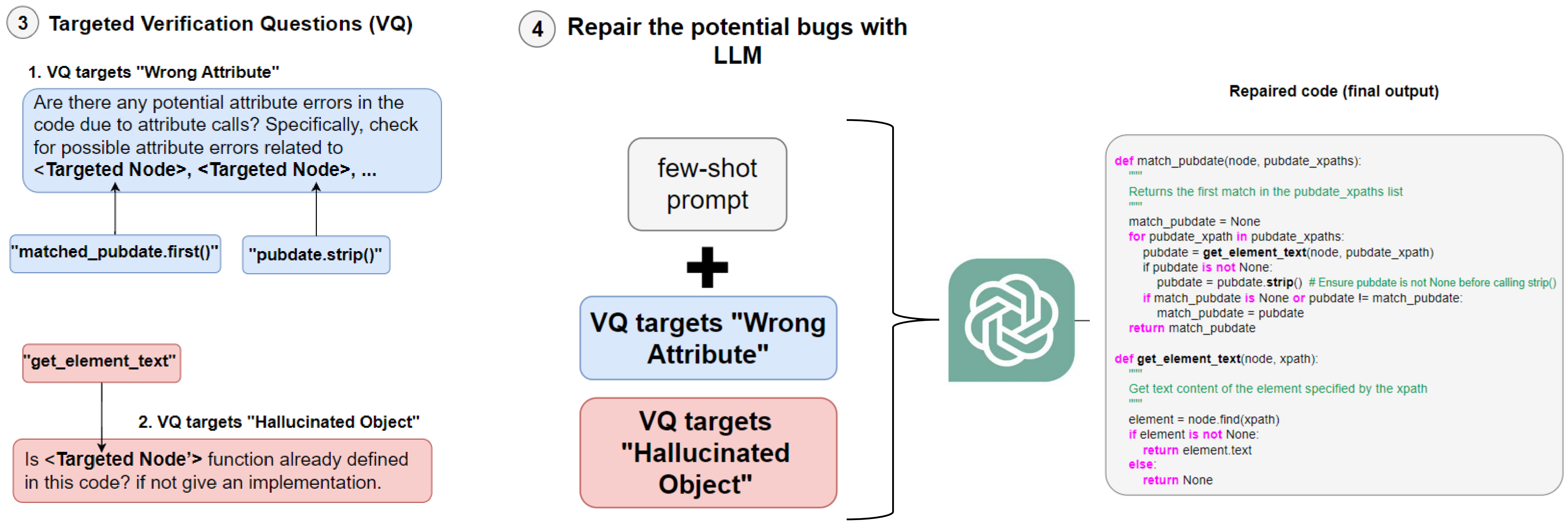
Targeted Nodes:

FuncCall -> Name: "get_element_text"

Attribute -> Name: "pubdate" , attr: "strip"
Attribute -> Name: "matched_pubdate" , attr: "first"

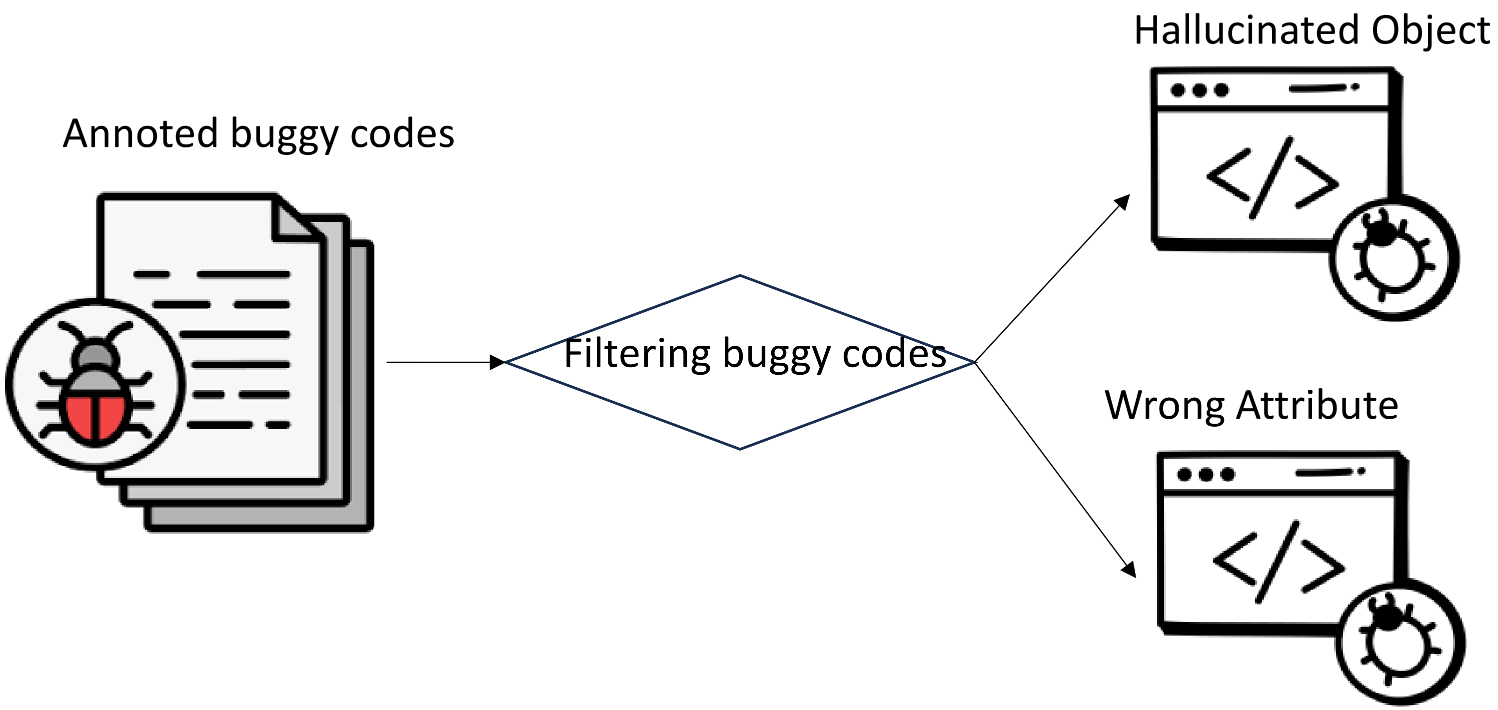
To localize the potential bugs, the method walks through the AST of the initial LLM-generated code **and collects features on some targeted nodes that may trigger specific types of errors.**

Generation of Verification Questions and Repair



The verification questions allowed the model **to repair the wrong attribute** <match_pudate.first()> and **implement the missing** <get_element_text()> function.

Evaluation




```
def match_pubdate(node, pubdate_xpaths):
    """
    Returns the first match in the pubdate_xpaths list
    """
    matched_pubdate = None
    for pubdate_xpath in pubdate_xpaths:
        pubdate = get_element_text(node, pubdate_xpath)
        if matched_pubdate is None or pubdate.strip() !=
matched_pubdate.first():
            matched_pubdate = pubdate
    return matched_pubdate
```

Evaluation

RQ1: Do the chain of VQs repair the bugs in LLM-generated code ?

RQ2: Can VQs introduce new bugs in LLM-generated code?

# of tasks	# of samples (hallucinated and wrong attribute)	# of Correct codes of these tasks
36	61	54

RQ1: Do the chain of VQs repair the bugs in LLM-generated code ?

		+ VQ
Runnable cases	10.03 %	35.75 %
Attribute errors	17.3 %	6.04 %
Name errors	15.13 %	4.175%
Other errors	25.54 %	22.03 %

Verification questions improved the performance of LLMs !

RQ2: Can VQs introduce new bugs in LLM-generated code?

Error types	Average number of samples
Correct code to Attribute errors	0.2%
Correct code to Name errors	0.2%
Correct code to Assertion errors	2 %
Correct code to Other errors	3.8%

Chain of VQs may introduce some bugs in correct code !

Rephrasing the questions of chain of VQs does not introduce high variability in the results

DeepCodeProbe: Towards Understanding What Models Trained on Code Learn

VAHID MAJDINASAB, Polytechnique Montreal, Canada
AMIN NIKANJAM, Polytechnique Montreal, Canada
FOUTSE KHOMH, Polytechnique Montreal, Canada

Determine if DL models trained on code learn programming language syntax.

- Assess if models retain syntax in latent space.
- If not, investigate alternative learned features.

Guidelines & Best Practices

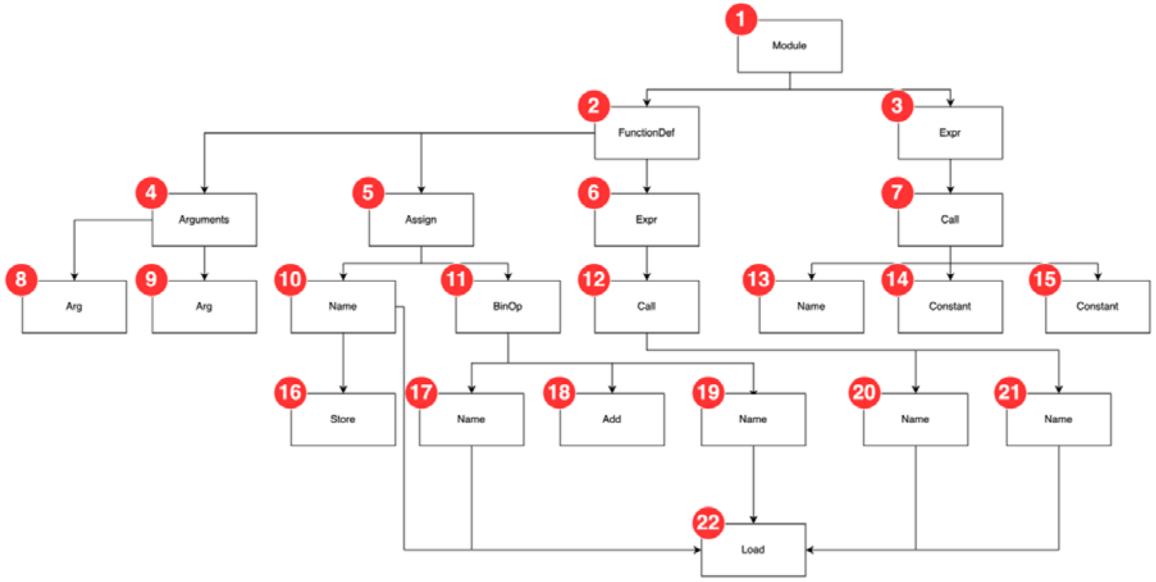
- Establish effective training strategies.
- Identify common pitfalls to avoid.

Algorithm 2 DeepCodeProbe

```
procedure ExtractDCU(SCRIPT)                                          ▹ Extract DCU tuple from code
    Input: SCRIPT                                                     ▹ The input code script
    Output: DCU                                                       ▹ The extracted DCU tuple
    if MODEL_INPUT_TYPE = AST then                                    ▹ If model uses ASTs
        SCRIPT_AST ← ConstructAST(SCRIPT)                             ▹ Construct AST
    else if MODEL_INPUT_TYPE = CFG then                               ▹ If model uses CFGs
        SCRIPT_CFG ← ConstructCFG(SCRIPT)                             ▹ Construct CFG
    end if
    SCRIPT_DCU ← Tree2Tuple(SCRIPT_AST or SCRIPT_CFG)                 ▹ Convert AST/CFG to DCU tuple
    return SCRIPT_DCU
end procedure
procedure ProbeModel(TRAINING_DATA, MODEL)                            ▹ Probe trained model
    Input: TRAINING_DATA                                              ▹ Code used to train MODEL
    Input: MODEL                                                      ▹ Trained model
    Output: PROBING_RESULTS                                           ▹ Probing results
    Probe ← InitializeProbe()                                         ▹ Initialize probe
    for code ∈ TRAINING_DATA do
        DCU ← ExtractDCU(code)                                        ▹ Extract DCU tuple
        model_embeddings ← MODEL(code)                                ▹ Get model embeddings
        Predicted_DCU ← Probe(model_embeddings)                       ▹ Probe model
        accuracy_d, accuracy_c, accuracy_u ← Compare(DCU, Predicted_DCU)   ▹ Compare predictions
        PROBING_RESULTS ← PROBING_RESULTS ∪ {accuracy_d, accuracy_c, accuracy_u}   ▹ Store results
    end for
    return PROBING_RESULTS
end procedure
```

Probing Approach

- Models under study are trained on AST/CFG.
- Models under study are not large.
- So we annotate the AST/CFG level by level, from left to right.
- We then create a mapping from the annotated AST/CFG which severely reduces its size.
- Our mapping is bi-directional. AST/CFG can be re-constructed from the mapping.
- We represent the mapping with a ***(d, c, u)*** tuple.




```
d = [ 1, 2, 3, 4, 5, 6, ....., 22 ]
c = [
        (2, 3) # children of node 1.
        (4, 5, 6) # children of node 2,
        (7) # children of node 3,
        (8, 9) # children of node 4,
        .......
    ]
u = [
        1 # Label of the ''Module'' node according to the model under study,
        54, # Label of the ''FunctionDef'' node according to the model under
            study,
        32, # Label of the ''Expr'' node according to the model under study,
        ......
    ]
```

Predicting the < d, c,u > tuple from the representations extracted from the hidden layers of the model, given a code snippet as input, indicates that the model is capable of representing the syntax of the programming language in its latent space

Models Analyzed

We study 4 models, across two different tasks.

- Code Clone Detection (CCD):
  - AST-NN (Encoder/Decoder)
  - FuncGNN (Graph Neural Network)
- Code summarization and comment generation:
  - Summarization-TF (Seq2Seq)
  - CodeSumDRL (Encoder/Decoder with Attention)
- Each of the models works on AST/CFG extracted from code.
- Each model has a different popular architecture.

RQ1 - Can models retain syntax in their latent space?

Result of DeepCodeProbe's accuracy on recovering Syntactic Information

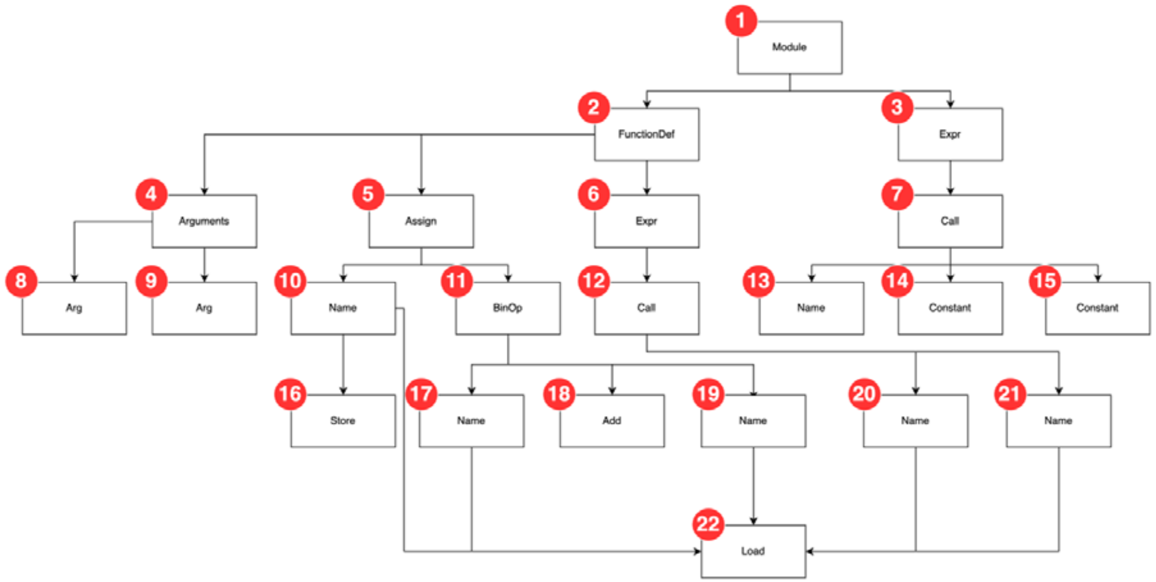
Model	Task	Programming Language	Accuracy-D(%)	Accuracy-C(%)	Accuracy-U(%)
AST-NN	CCD	C	8.65	8.63	8.65
		Java	8.33	8.09	8.65
FuncGNN	CCD	Java	43.36	98.51	39.26
SummarizationTF	Code Summarization	Java	13.83	14.79	14.79
CodeSumDRL	Code Summarization	Python	41.60	33.92	28.08

- Our probe shows that FuncGNN focuses on the connections between nodes in the CFG to detect clones.
- None of the models under study are capable of representing the full syntax of the programming language in their latent space

RQ2 - If not the syntax, then what are the models learning?

Probing for proxies

- Instead of looking for complete syntax information, we probe for more general information.
- Such information cannot be used to reconstruct the AST but it is extracted from the AST nonetheless.
- Instead of constructing a ***(d, c, u)*** tuple, we construct a ***(c, u)*** tuple:
    - C: whether the node has any children (it is connected to another node or not)
    - U: the general label of a node (instead of the fine-grained label we were using)




```
c = [
        1 # node 1 has children.
        1 # node 2 has children,
        1 # node 3 has children,
        .......
        0 # node 8 has no children
        0 # node 9 has no children
        .....
    ]
u = [Module, FunctionDef, Expr, Arguments, ....]
```

Listing 2. Generated $< c, u >$ tuple for the AST in Figure 2

RQ2 - If not the syntax, then what are the models learning?

Result of DeepCodeProbe's accuracy on recovering Syntactic Information

Model	Task	Programming Language	Accuracy-C(%)	Accuracy-U(%)
AST-NN	CCD	C	99.17	62.11
		Java	97.50	61.25
SummarizationTF	Code Summarization	Python	73.64	60.97
CodeSumDRL	Code Summarization	Python	43.21	32.03

- We observe **significant increase in syntactic information** recovered from the models
- Therefore, **the models DO retain syntactic information** from their training data.
- Even though it is not the complete syntax, **the models learn abstracts of the syntax of the programming language.**
- So, for software maintenance tasks, **we may not need large models.**
- **We do not need the models to learn the entire syntax** of the programming language.

Lessons Learned

Our probing shows that using AST of codes for training small models is beneficial:

- **Even though the models do not explicitly learn the syntax**, they learn an abstraction from it.
- Therefore, **using representations that explicitly encode the syntax of code** can result in models that are smaller, less resource intensive and capable.

Statement level vs word level tokenization:

- Except for FuncGNN, each model uses some form word level tokenization for encoding information from the ASTs.
- FuncGNN shows that **statement level tokenization can be more useful** in pushing the model to focus on syntactic information.

Code clones are useful in testing the model outside of its trained task:

- As code clones are codes that are similar to each other to varying degrees, regardless of the task the model is trained for, **it should display similar performance and encoding for similar codes.**

Lessons Learned

Efficacy of Syntactical Representations: Models don't need to fully learn syntax to perform well on software maintenance tasks.

- Models can learn syntax abstractions from syntactically valid code representations.
- Smaller, effective models can be trained using artifacts from code.
- Benefits include reduced model size and improved efficiency without sacrificing performance.

Tailoring Data Representations:

- AST-NN: Uses smaller sub-trees and Word2Vec for compact, effective code clone detection.
- FuncGNN: Statement-level tokenization in CFGs for detailed representation.
- SummarizationTF & CodeSumDRL: Treat code summarization as translation using ASTs for efficient performance.

Recommendation: Use syntactic representations (AST/CFG) for training models on code.

Lessons Learned

Enhancing Model Reliability:

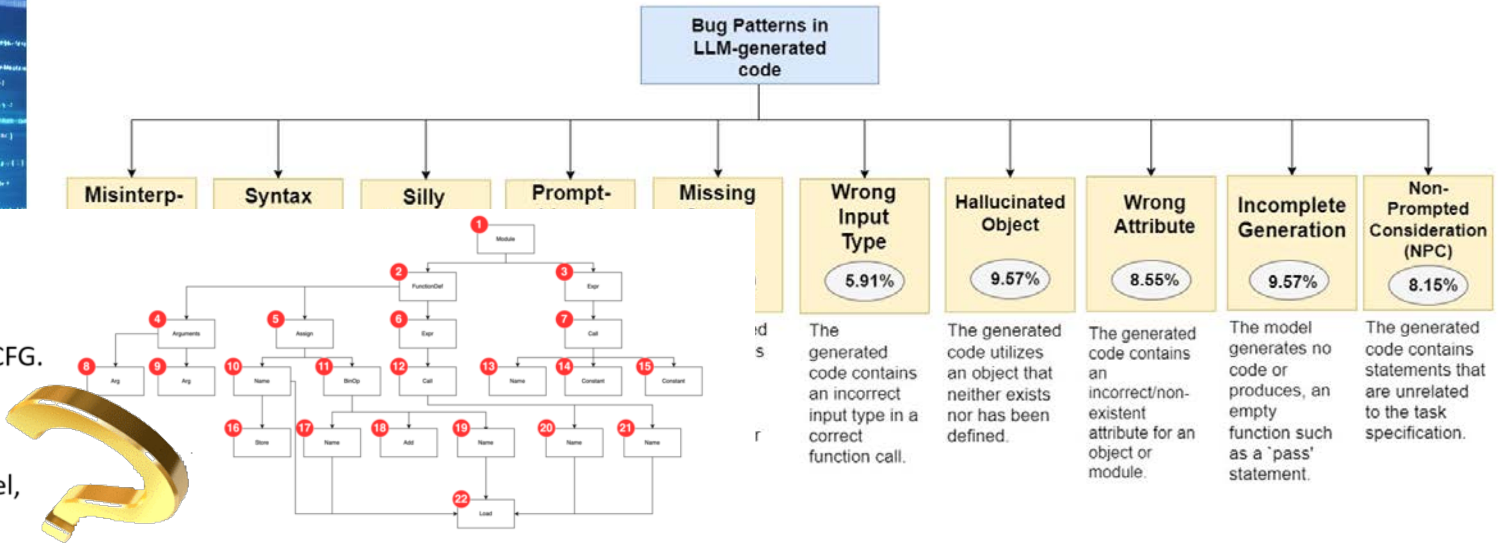
Interpretability:

- Probing reveals model decision-making processes.
- Helps identify errors and refine models.
- Smaller models (RNNs, encoder-decoder, seq2seq) are computationally efficient and interpretable.

Contrast with LLMs

- LLMs are resource-intensive, prone to hallucinations, and lack interpretability.
- Smaller models offer reliability and efficiency, making them suitable for software maintenance tasks.

We are entering in an Era of AI-assisted Software Engineering



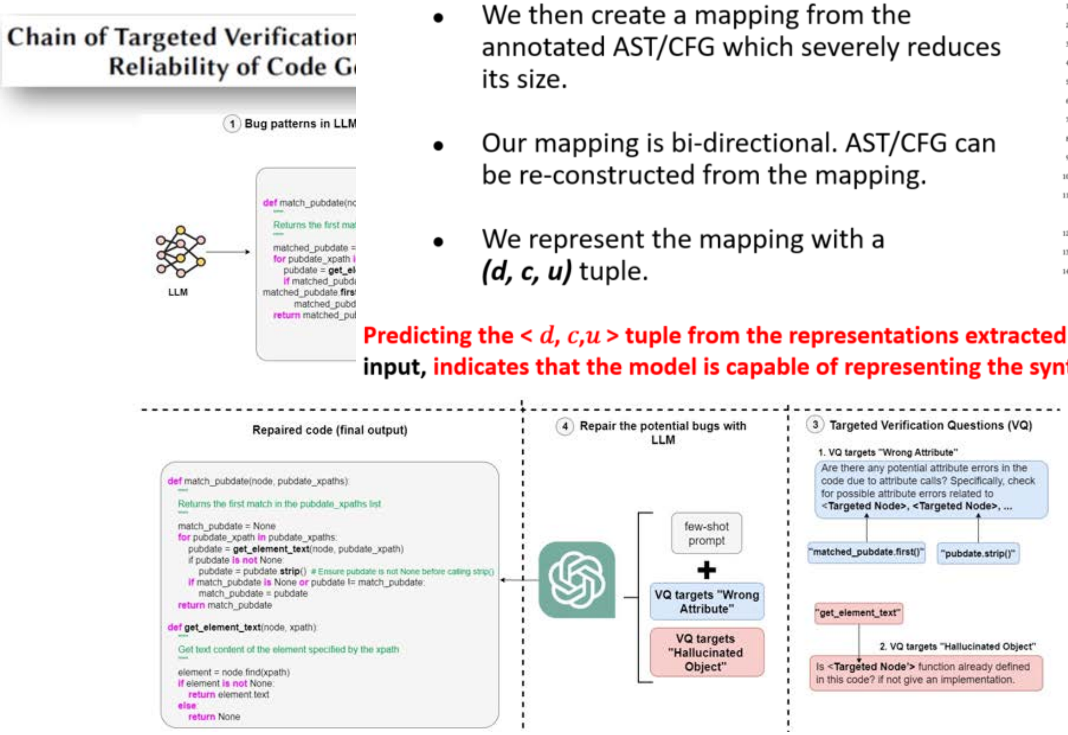
Probing Approach

- Models under study are trained on AST/CFG.
- Models under study are not large.
- So we annotate the AST/CFG level by level, from left to right.
- We then create a mapping from the annotated AST/CFG which severely reduces its size.
- Our mapping is bi-directional. AST/CFG can be re-constructed from the mapping.
- We represent the mapping with a *(d, c, u)* tuple.

```
d = [ 1, 2, 3, 4, 5, 6, ......, 22 ]
c = [
        (2, 3) # children of node 1.
        (4, 5, 6) # children of node 2,
        (7) # children of node 3,
        (8, 9) # children of node 4,
        .......
    ]
u = [
        1 # Label of the ''Module'' node according to the model under study,
        54, # Label of the ''FunctionDef'' node according to the model under
            study,
        32, # Label of the ''Expr'' node according to the model under study,
        ......
    ]
```

Predicting the < d, c, u > tuple from the representations extracted from the hidden layers of the model, given a code snippet as input, indicates that the model is capable of representing the syntax of the programming language in its latent space

47



...n CoderEval tasks

	ChatGPT	ChatGPT + VQ
	10.03 %	35.75 %
	17.3 %	6.04 %
Name errors	15.13 %	4.175%
Other errors	25.54 %	22.03 %

Verification questions improved the performance of LLMs !