AI for Software Engineering

Jibesh Patra

Week 14 - CS20202 - Spring 2025 - IIT Kharagpur

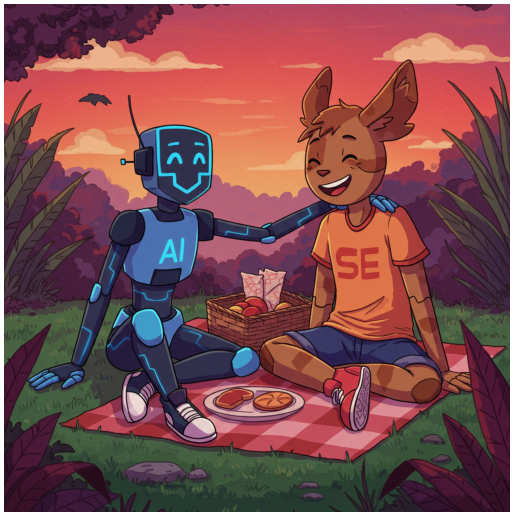
Materials adapted from “A Survey on Large Language Models for Software Engineering - Zhang et. al.” and papers of respective authors.

Intersection of Artificial Intelligence (AI) and Software Engineering (SE)

AI4SE vs SE4AI

- **AI4SE**: Using AI to enhance Software Engineering tasks which helps in building robust and reliable systems.
- **SE4AI:** Using Software Engineering techniques to build robust AI systems.

This lecture: **AI4SE**



Popular Public Applications

- Code generation - GitHub Copilot
- Bug Detection - DeepCode
- Automated Testing - Testim
- Code Documentation - Swim (documentation + modernization)

Usages of AI in Software Development

mint

100 15256.75 -132.25 (-0.86%) ↓ | India VIX 21.43 +0.99 (4.83%) ↑ | S&P BSE 200 10119.02 -56.03 (-0.55%

Home Latest News Markets News Premium Companies Money Education Te

Over 25% of Google software is written by AI, says CEO Sundar Pichai. Is it a threat to engineers?

During an earnings call, Google CEO Sundar Pichai disclosed that more than 25 per cent of new code is generated by AI. However, they are later reviewed by engineers.

Riya R Alex

Published • 30 Oct 2024, 07:37 PM IST

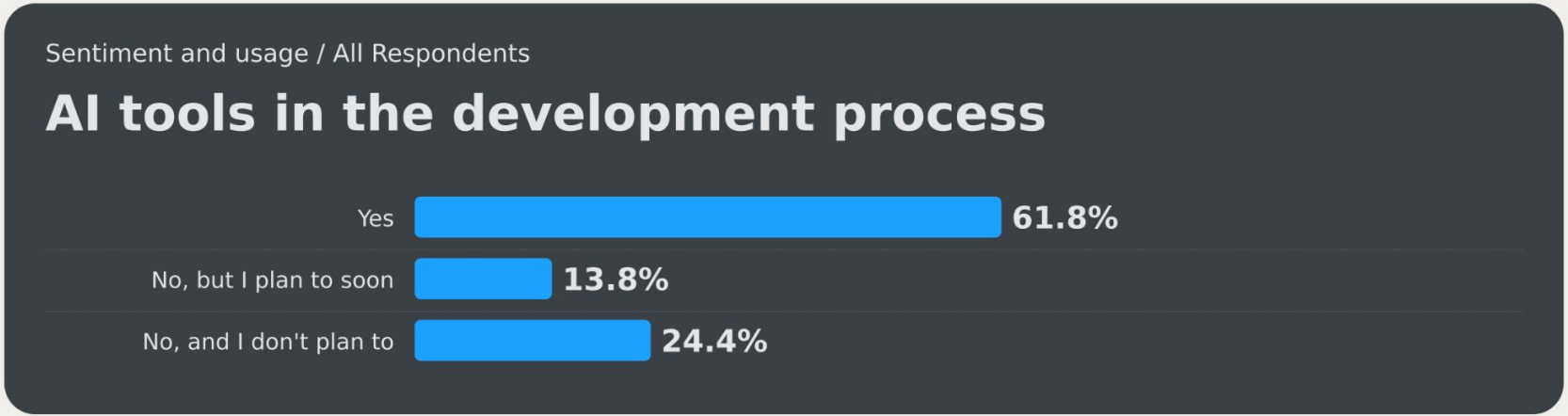
Home • Technology • Meta's AI can replace mid-level coders in 2025: Zuckerberg

Meta's AI can replace mid-level coders in 2025: Zuckerberg

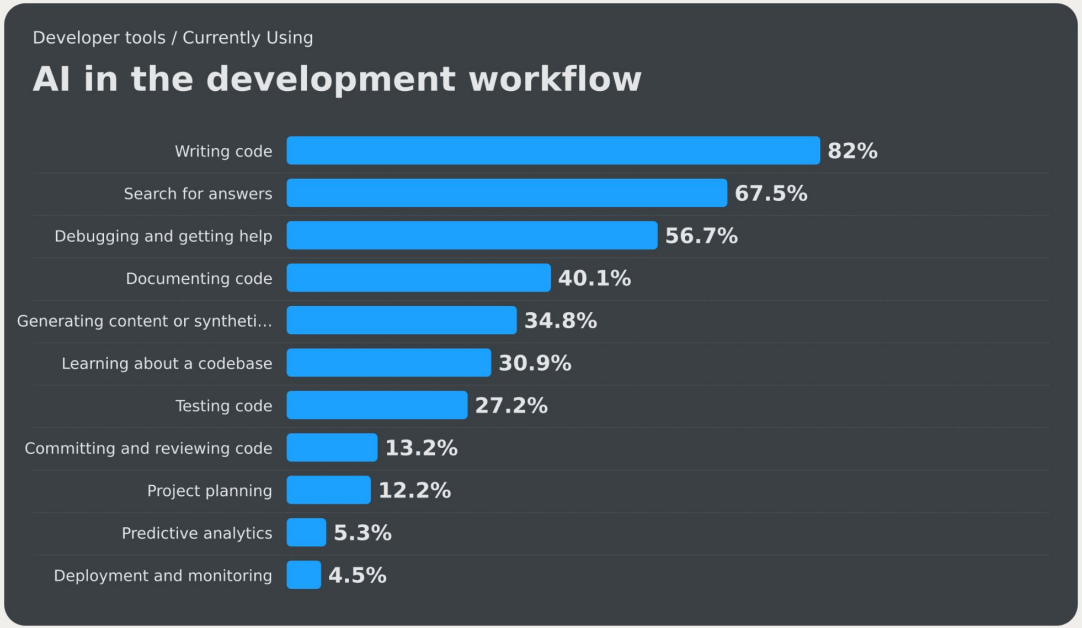
1805 Views | 14 Jan 2025, 04:12 PM | Gaurav Sharma

He went on also to explain that despite these AI systems being expensive to run at first, it is predicted that their performance will improve and some of them will be able to take a huge percentage of code-writing jobs that engineers have been doing.

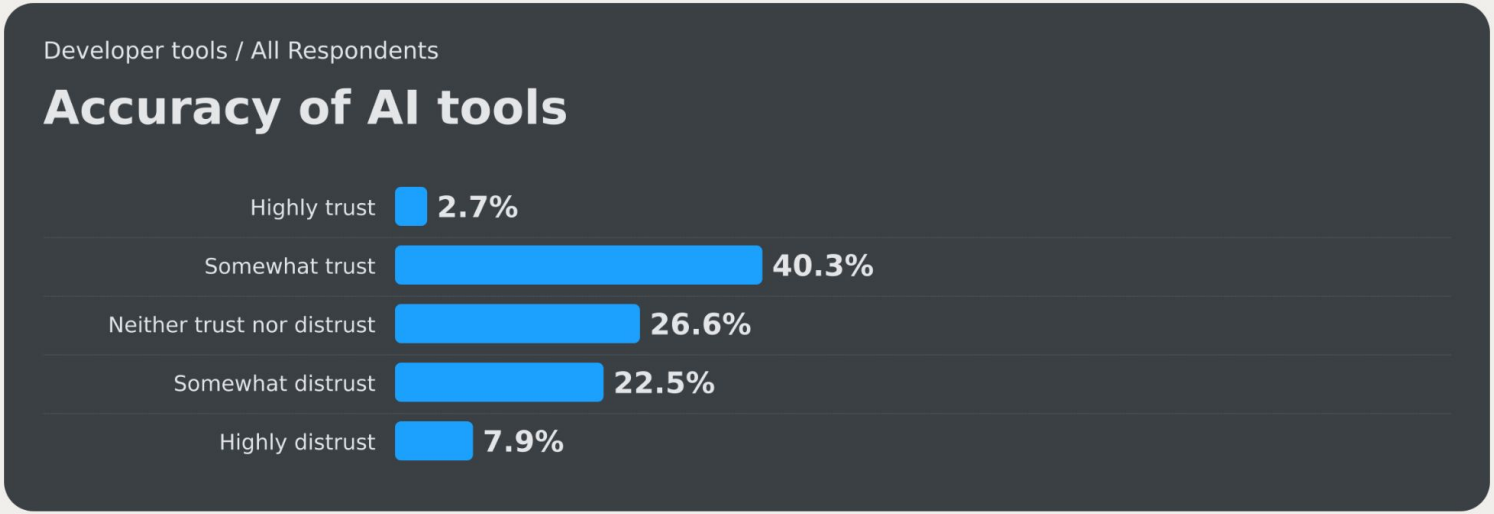
Opinion of Developers – Do you use AI in Development Process?



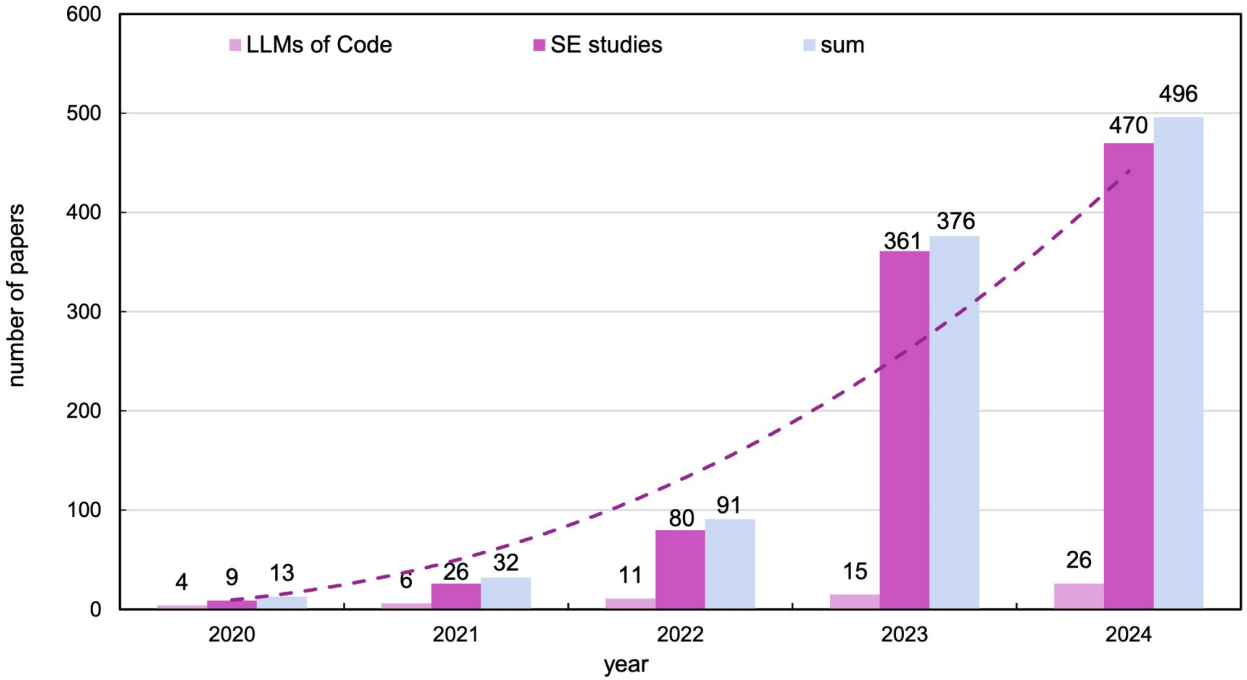
Opinion of Developers – Part of Development Process AI used?



Opinion of Developers – How much do you Trust AI Output?



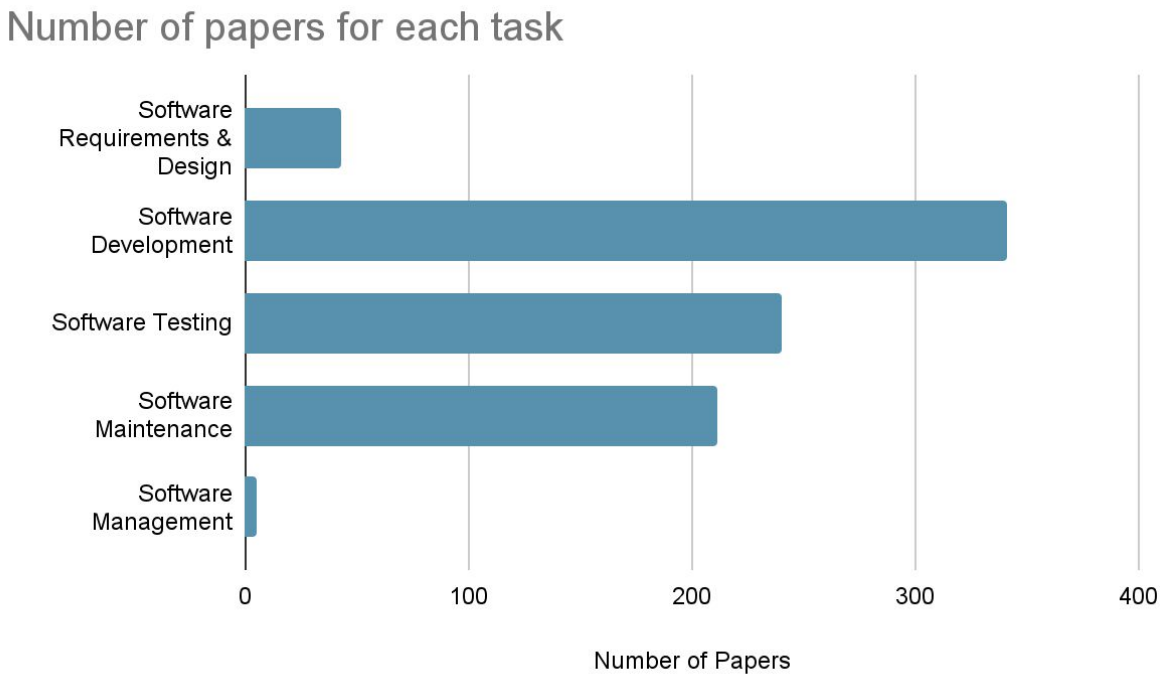
Large Language Models (LLMs) for Code



Papers over the years

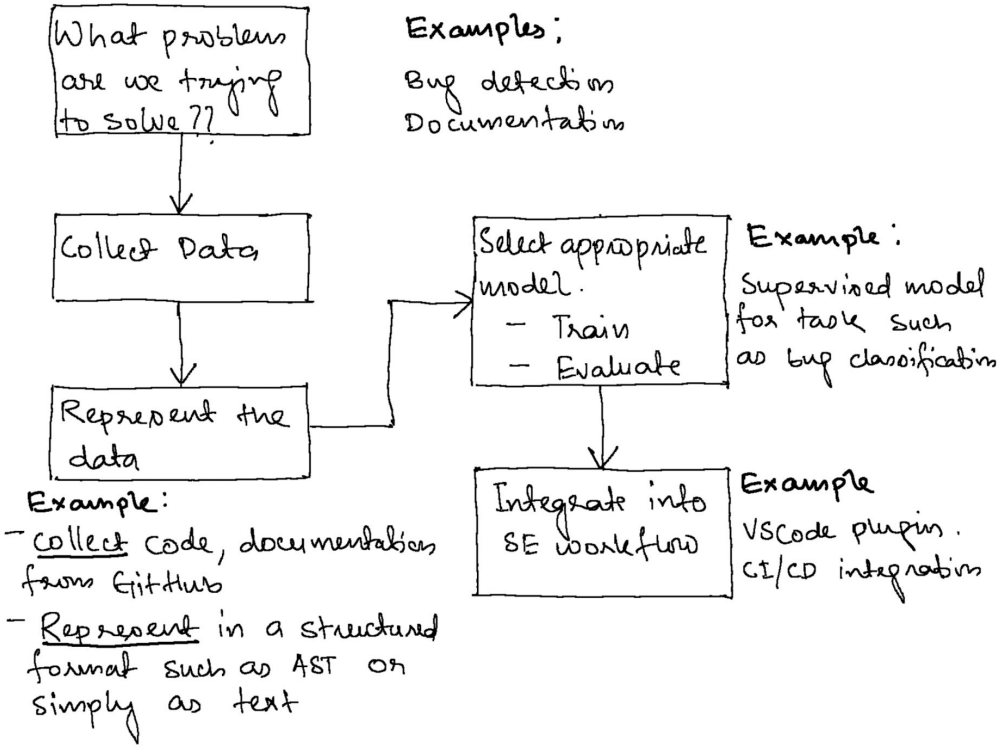
A Survey on Large Language Models for Software Engineering - Zhang et. al.

Downstream Tasks



A Survey on Large Language Models for Software Engineering - Zhang et. al.

General Workflow of AI4SE Systems



Typical LLM4SE

1. Collect data
2. Pre-train
3. Fine-tune for specific tasks
4. Integrate into SE workflows

Data Collection

Data collection is the first step of the system:

- Source code from open source code repositories such as GitHub, GitLab.
- Bug reports, Issues, Commits.
- Documentations, Code reviews.
- Example: The Stack Dataset
  - Uses GitHub archive to extract the dataset
  - Contains data from more than 350 programming languages
  - 6 TB of code data

Pre-training

- Trained on massive corpora of code and natural language obtained from
  - Public repositories such as GitHub, GitLab
  - Q&A websites frequently visited by developers such as Stack Overflow, Reddit
  - Documentations such as API docs
- Quality of the data is very important.
- Example training objectives:
  - Causal language modeling
  - Masked language modeling
  - Replaced token detection

Fine-tuning

Refine the pre-trained language model for particular tasks by training it further using specific datasets.

Example: CodeBERT, a language model is fine-tuned for natural language code search.

Integration Into SE Workflows

Integrate LLM for practical software engineering tasks.

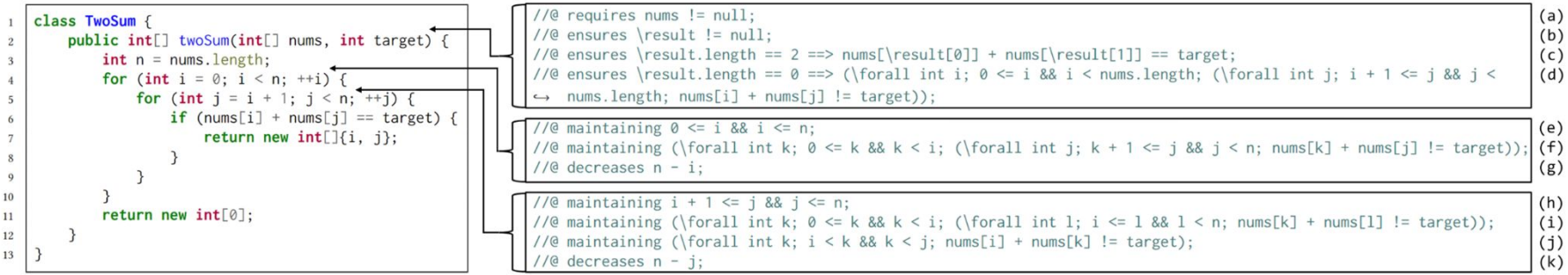
Example: GitHub Copilot has been integrated as a plugin for VS Code and assists developers with AI code completion, Natural language chats etc.

Software Requirement & Design

SpecGen

SpecGen: Automated Generation of Formal Program Specifications via Large Language Models by *Lezhi Ma et al.* (2025)

- Program specifications encompass precise statements that describe the intended or actual behaviors of a particular program.



Program and corresponding specifications generated by *SpecGen*.

SpecGen

- **Motivation**: Existing automated program specification generation rely on manually created templates resulting in simple and trivial specifications.
- **Approach**:
    a. Given input code and a prompt, LLM generates a specification and verified by verifier.
    b. This is refined further using conversation.
    c. For the the *still* failing generated specs, mutate the code:
        - Take locations of code where LLM generates wrong spec.
        - Mutate the locations with various mutation operators.
    d. Check with the verifier and select the best specification.

Examples of LLM Applications

- **Requirement classification**: Categorization of software requirements into different classes or types e.g., functional or non-functional.
  - NoRBERT by Hey et al. (2020) fine-tunes an existing LM called BERT. Existing automatic classification performs poorly on unseen projects which is improved by the authors.
    - Evaluate on a dataset containing classes of requirement.
- **Requirement ambiguity detection**: Ambiguity (where the natural language description may be interpreted in more than one way) in software requirements may result in production of poor quality software.
  - TABASCO by Moharil et al. (2023) finds ambiguities using BERT to capture different meanings a word can have depending on its context within a requirement.

Software Development

ARCHCODE

ARCHCODE: Incorporating Software Requirements in Code Generation with Large Language Models by *Han et al.*

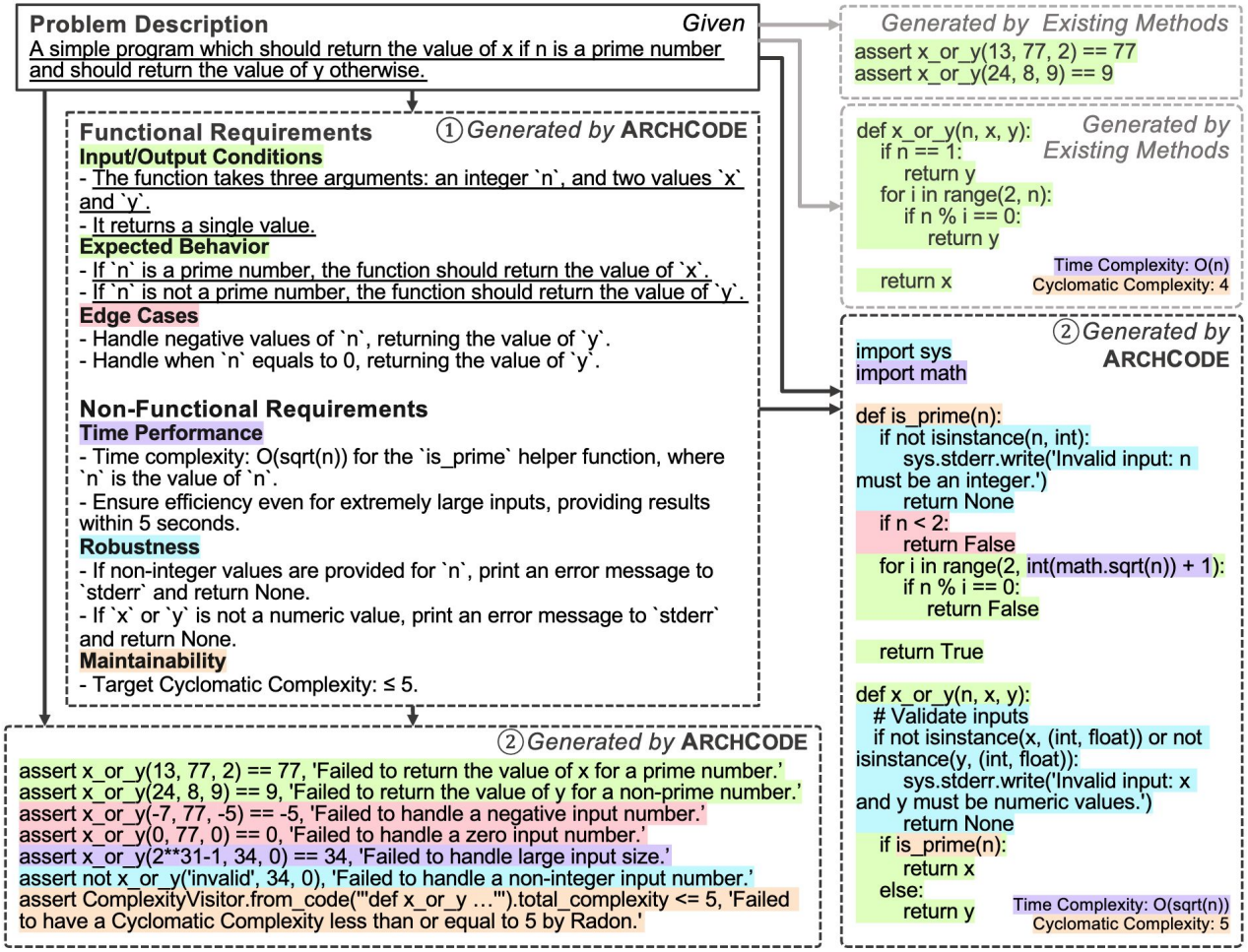
Motivation

- A natural language description include both functional and nonfunctional requirements.
- This can lead to LLM generated code that is functionally correct but violates certain requirements.

ARCHCODE

Approach

- Start with textual software requirements. Use LLMs to extrapolate unexpressed or implicit requirements and structure it.
- Incorporates structured requirement in the prompt and generate code and test cases.



Comparing Code and Test generation between ARCHCODE and existing approaches.

- Existing approaches directly generates code from description.
- In comparison ARCHCODE introduces structure.

Examples of LLM Applications

- **Code search**: Given a natural language query, retrieve functionally relevant code.
  - CodeRetriever by Li et al. (2022) performs code-text contrastive pre-training to learn function-level code semantics. This aids in better code search.
- **Code Translation**: Translating code in one programming language to another.
  - TransMap by Wang et al. (2023) detects semantic mistakes in code translated by ChatGPT using tests from source and translated program.
- **Code Summarization**: Use code as input and generates high-level natural language summaries.
  - ESALE by Fang et al. (2024) uses multi-task learning (unidirectional language modeling, masked language modeling, action word prediction) to improve code summarization.

Software Testing

Fuzz4All

Fuzz4All: Universal Fuzzing with Large Language Models by Xia et al. (2024)

Motivation

- Discover bugs using fuzzing.
- Traditional approach often target a specific language or features and can not easily applied to other languages or features.

Fuzz4All

Approach

- Use LLMs for input generation and mutation engines.
- Since LLMs are trained on multiple programming languages, they are applicable to multiple languages.

Examples of LLM Applications

- **Fault localization**: Identify specific locations in a software system where faults or bugs are present.
  - TROBO by Zhu et al., (2021) performs cross-project knowledge transfer of bug report and code file.
- **Vulnerability detection**: Identify potential security bugs in software systems.
  - VulLLM by Du et al., (2024) performs multi-task learning (vulnerability localization task, vulnerability interpretation) with LLMs to detect vulnerabilities.
- **Unit test generation**: Creating a set of test cases for testing the adequacy of software programs.
  - MuTAP by Dakhel et al., (2023) performs mutation testing to augment prompts to guide LLMs in generating test cases that can detect bugs.

THE END