From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

Song Wang

Associate Professor

EECS, York University, Canada

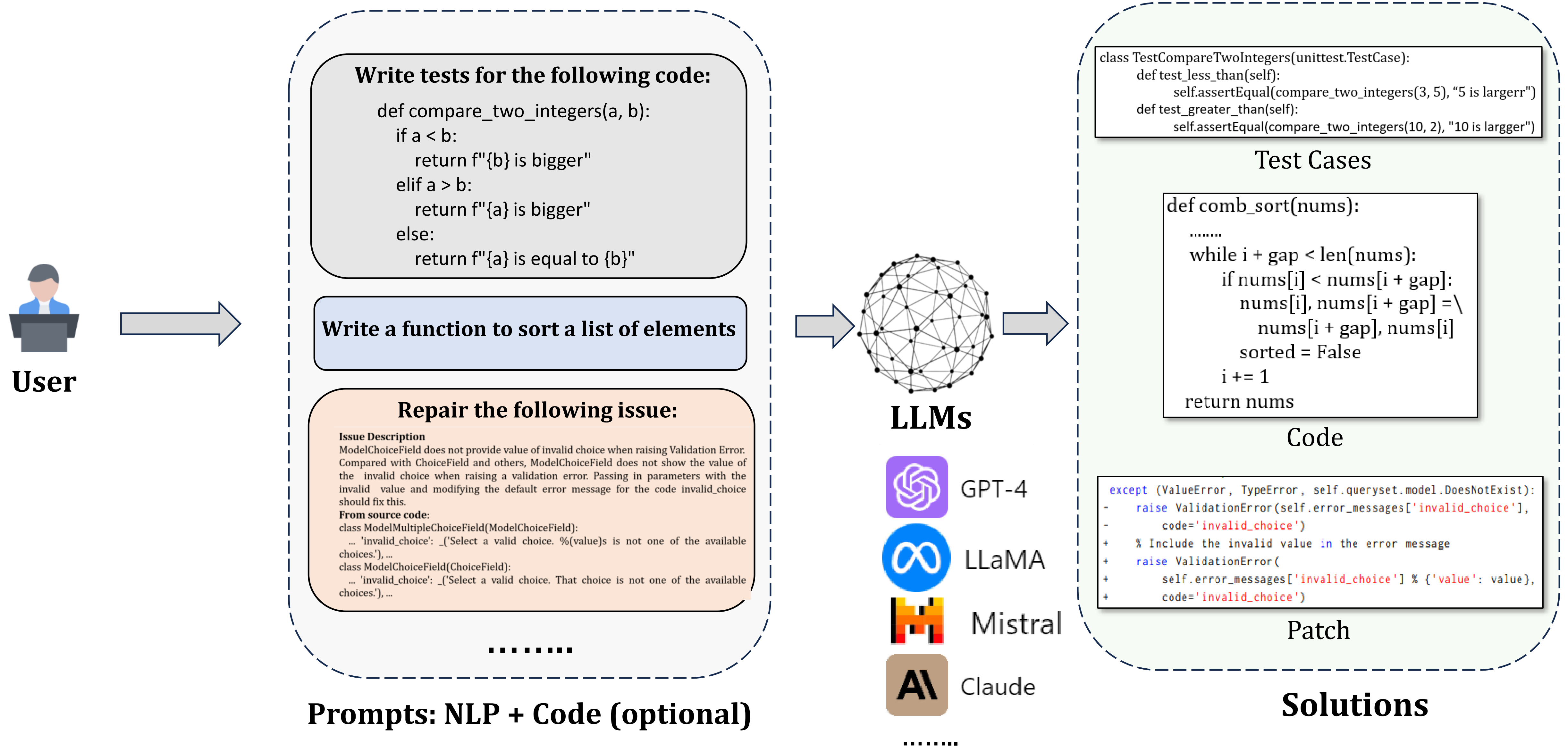
wangsong@yorku.ca

eecs.yorku.ca/~wangsong

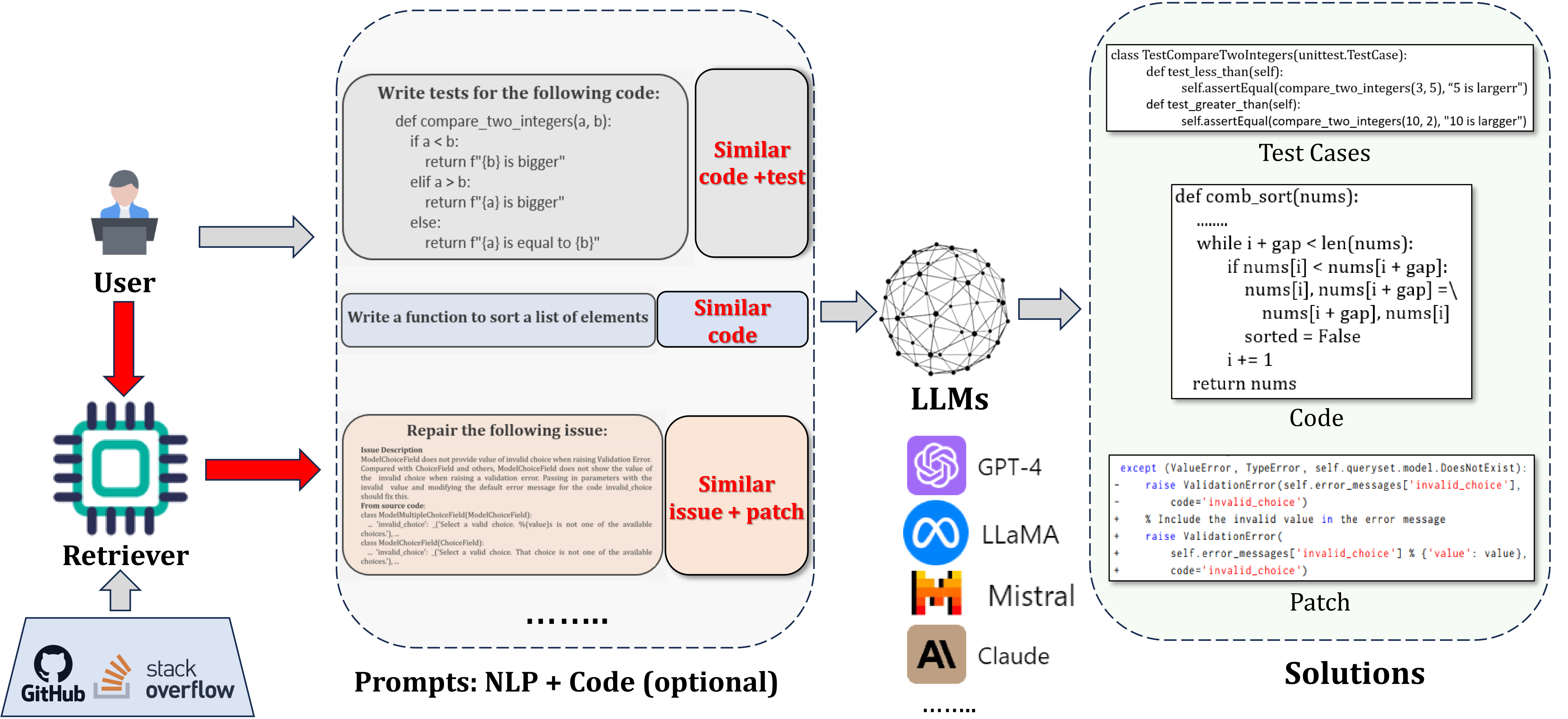
github.com/waooog

YORK UNIVERSITÉ UNIVERSITY

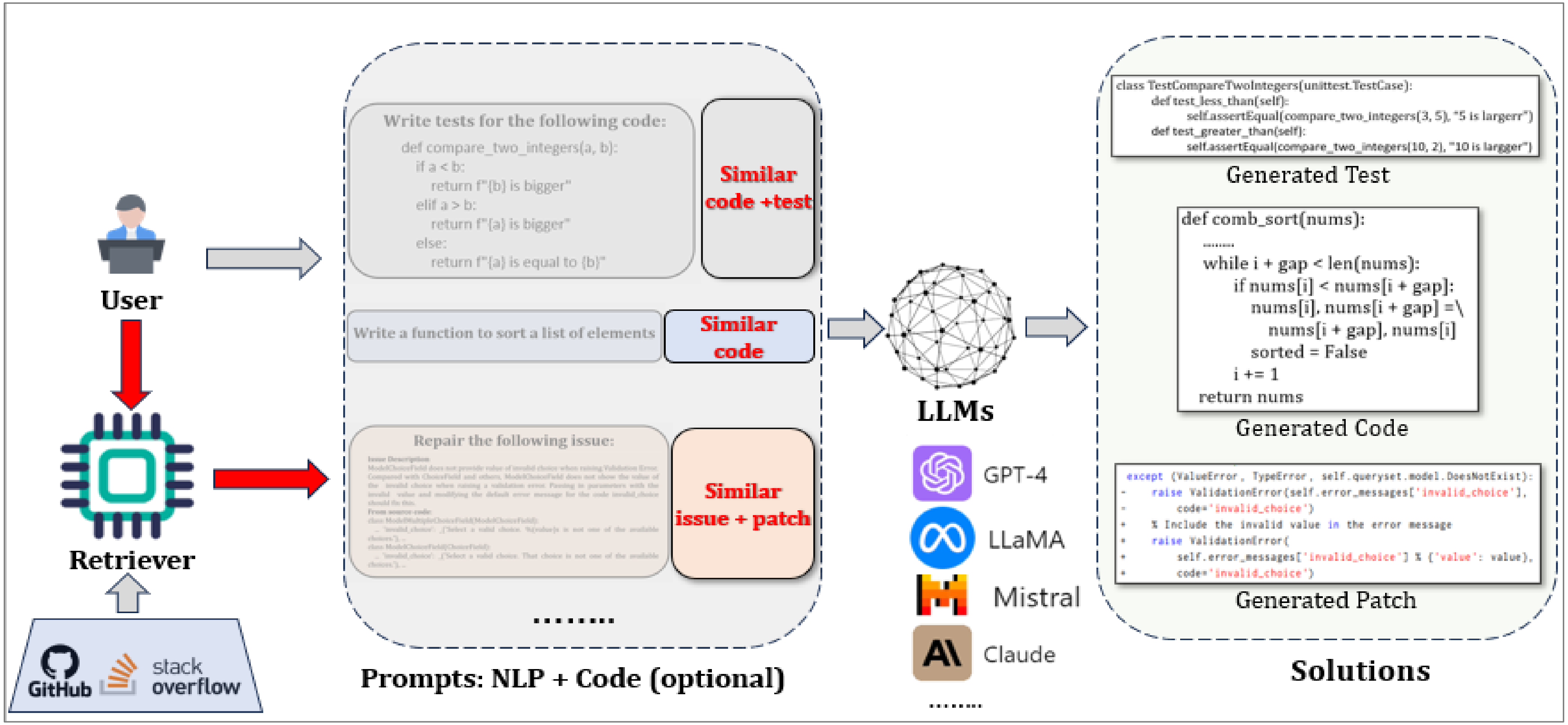
Automatic SE in the Era of LLM



LLM-based Automatic SE + RAG



Our recent work on advancing LLM-based SE



Enhancing Prompts via RAG

Enhancing Prompts via clarifying and optimization

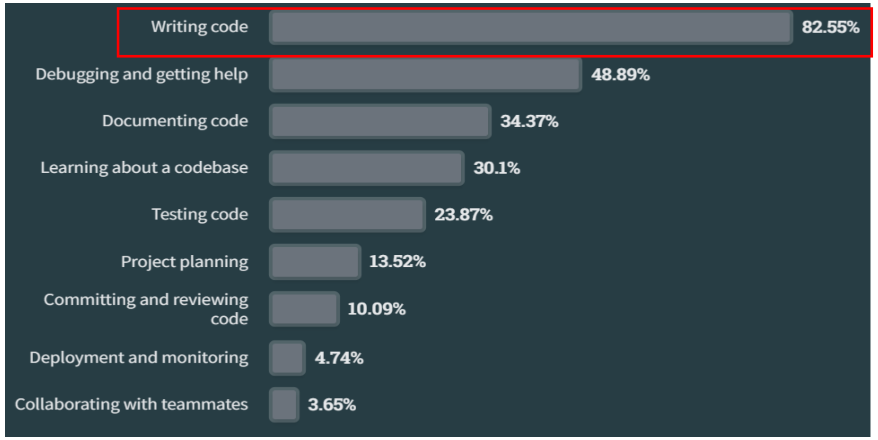
Fine-tuning small LLMs for SE tasks

Ensuring the reliability of benchmark data

Enhancing prompts

ClarifyGPT: Empowering LLM-based Code Generation with Intention Clarification

- **Auto Code Generation**
  - Converting user-provided natural language requirements to executable code
  - Improving software development efficiency



https://survey.stackoverflow.co/2023/#ai

User

Write a function to sort a list of elements.

NLP requirement

LLMs

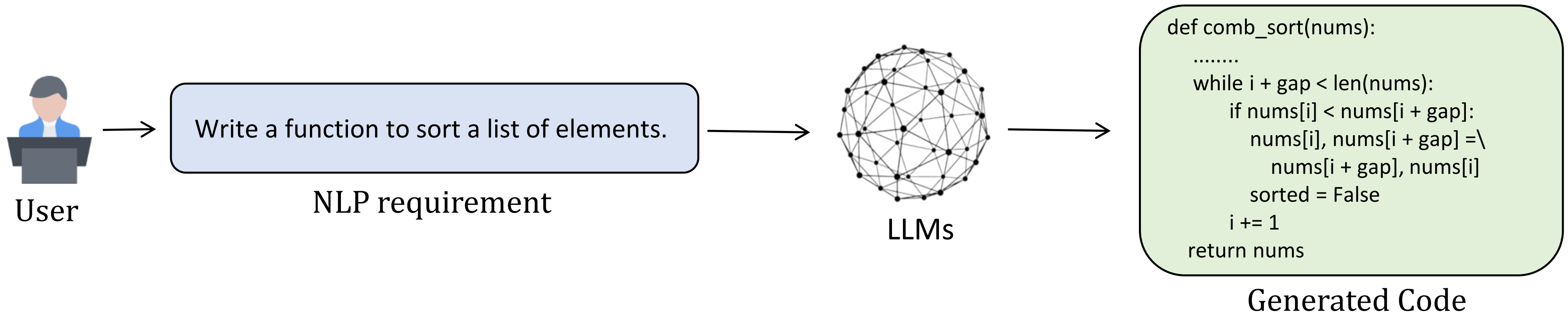
```
def comb_sort(nums):
    ........
    while i + gap < len(nums):
        if nums[i] < nums[i + gap]:
            nums[i], nums[i + gap] =\
                nums[i + gap], nums[i]
            sorted = False
        i += 1
    return nums
```

Code Solution

ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification
F Mu, L Shi, S Wang, Z Yu, B Zhang, CX Wang, S Liu, Q Wang
Proceedings of the ACM on Software Engineering 1 (FSE), 2332-2354

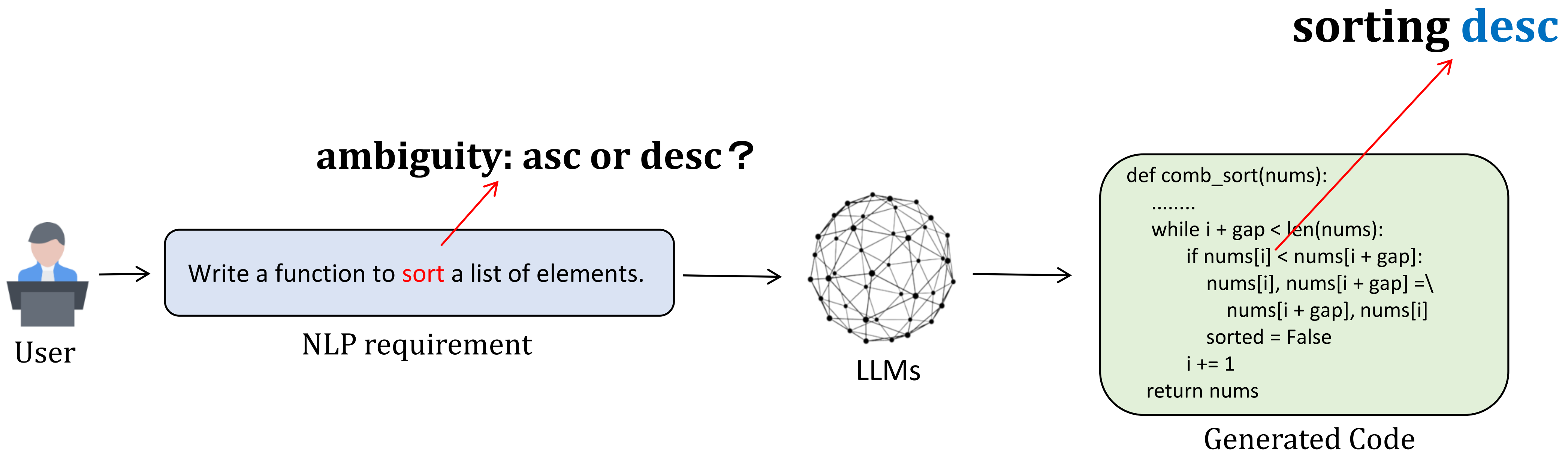
Challenges for LLM-based Code Generation

- Users often struggle to accurately express their requirements, leading to ambiguity in natural language descriptions
- LLMs lack machinimas to clarify the requirements



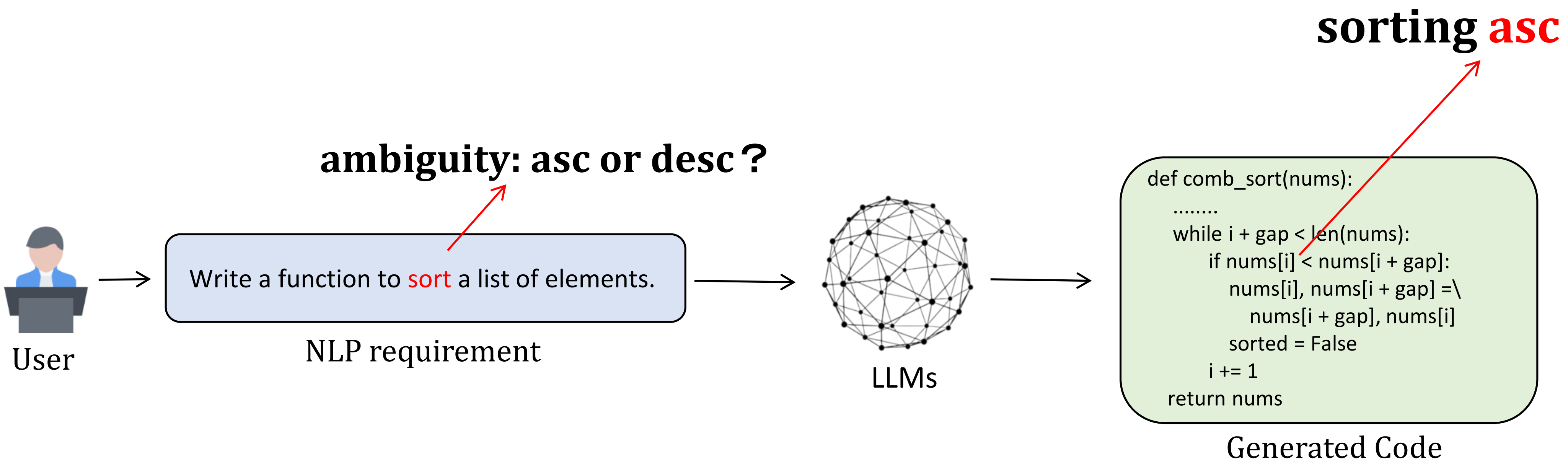
Challenges for LLM-based Code Generation

- Users often struggle to accurately express their requirements, leading to ambiguity in natural language descriptions
- LLMs lack machinimas to clarify the requirements



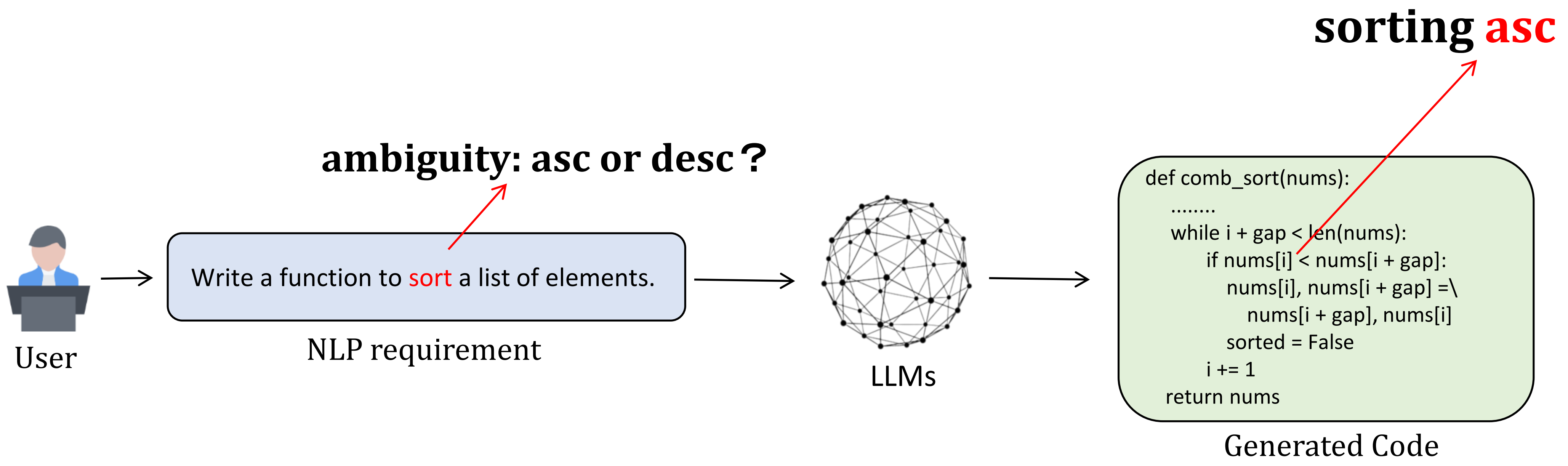
Challenges for LLM-based Code Generation

- Users often struggle to accurately express their requirements, leading to ambiguity in natural language descriptions
- LLMs lack machinimas to clarify the requirements



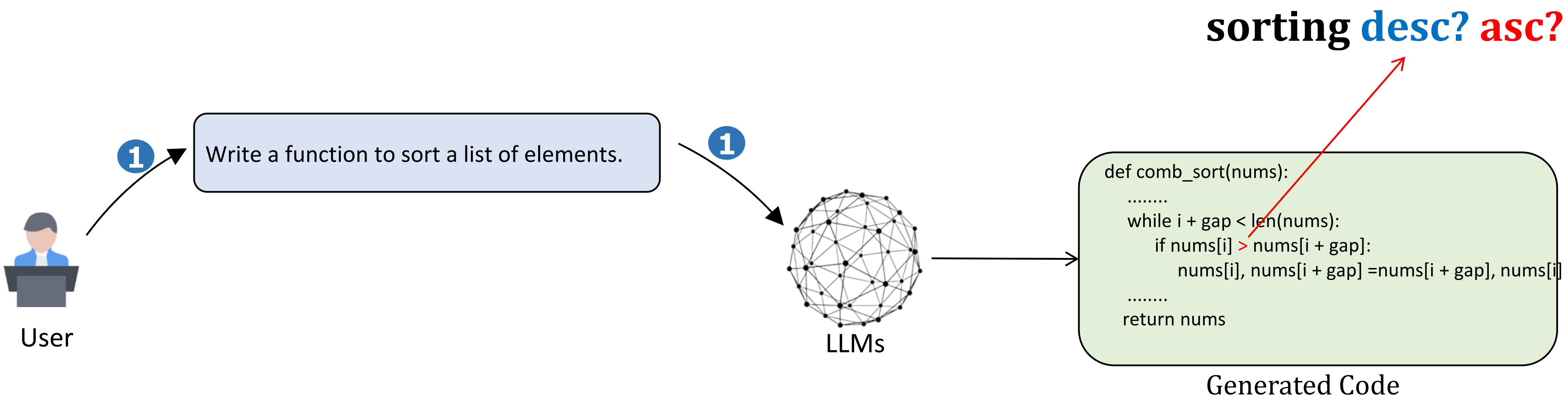
Challenges for LLM-based Code Generation

- Users often struggle to accurately express their requirements, leading to ambiguity in natural language descriptions
- LLMs lack machinimas to clarify the requirements

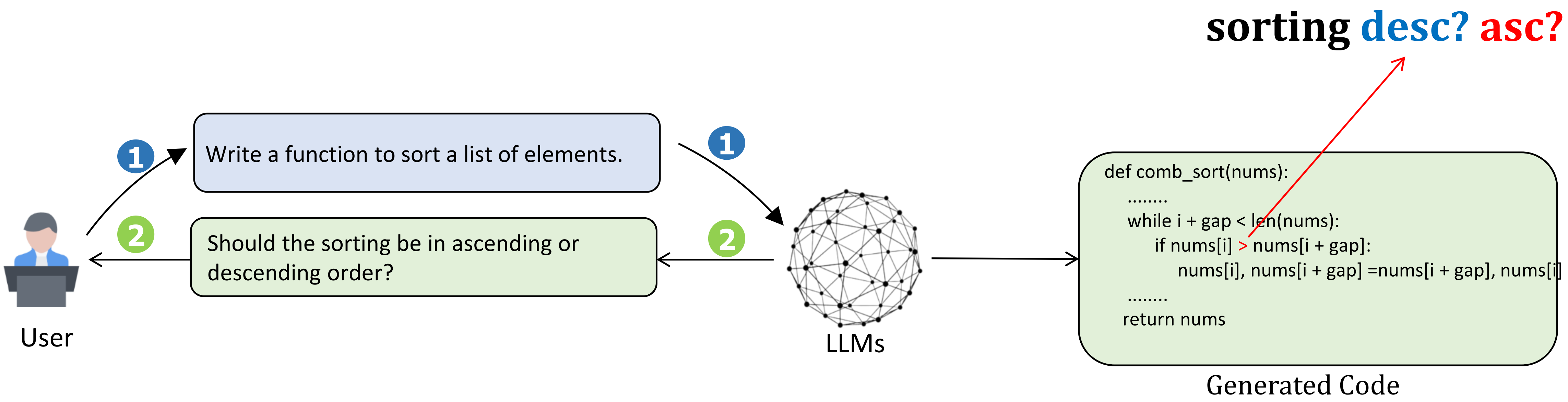


Observation: Ambiguous requirements often lead to semantic inconsistent code among different solutions generated by LLMs.

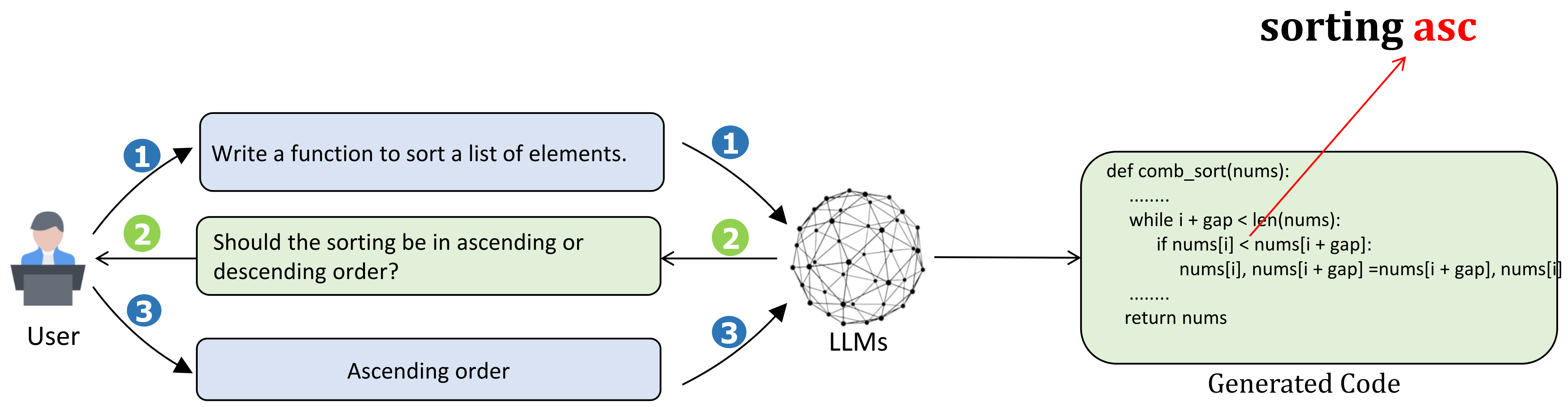
Solution: Human-in-the-loop Requirements Clarification



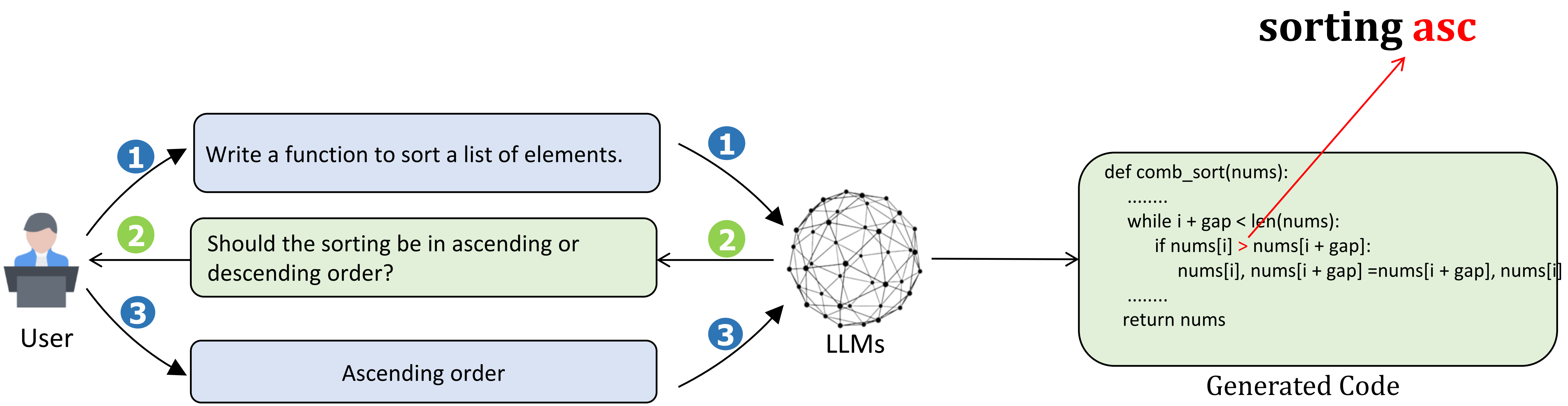
Solution: Human-in-the-loop Requirements Clarification



Solution: Human-in-the-loop Requirements Clarification

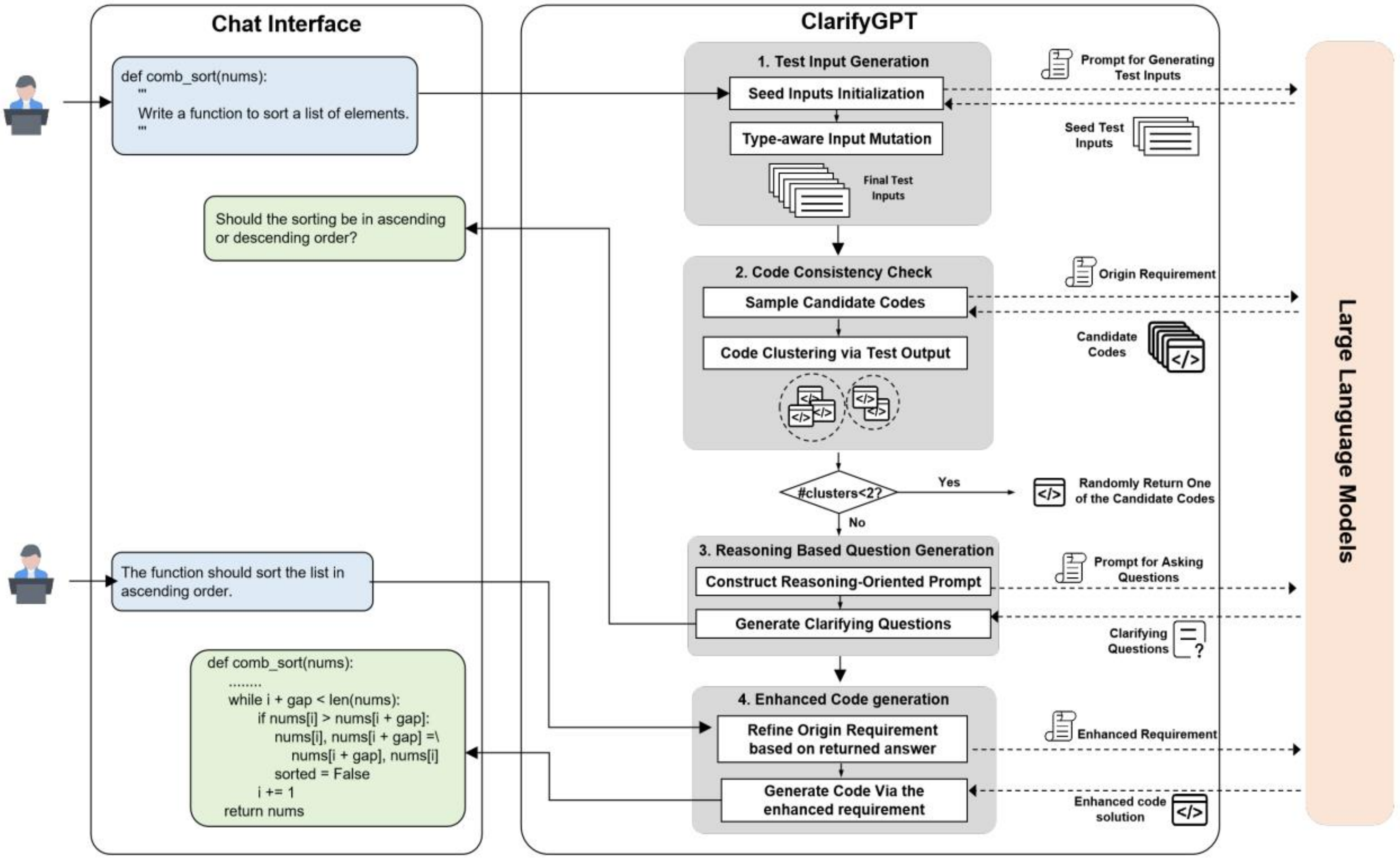


Solution: Human-in-the-loop Requirements Clarification



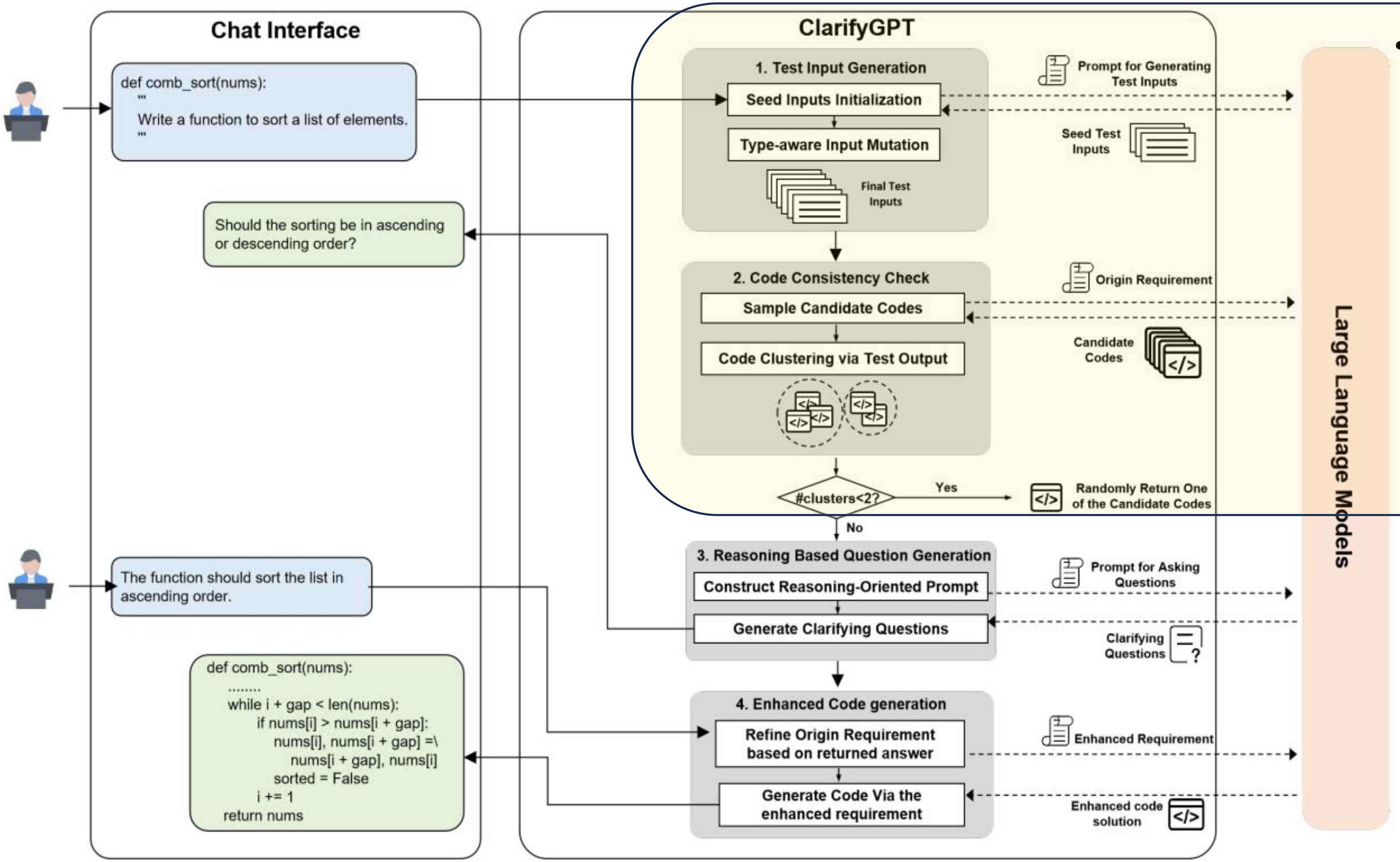
Goal: Empowering LLMs with the ability to identify and clarify ambiguous requirements would help LLMs generate accurate code.

Empowering LLM-based Code Generation with Intention Clarification



Overview of ClarifyGPT

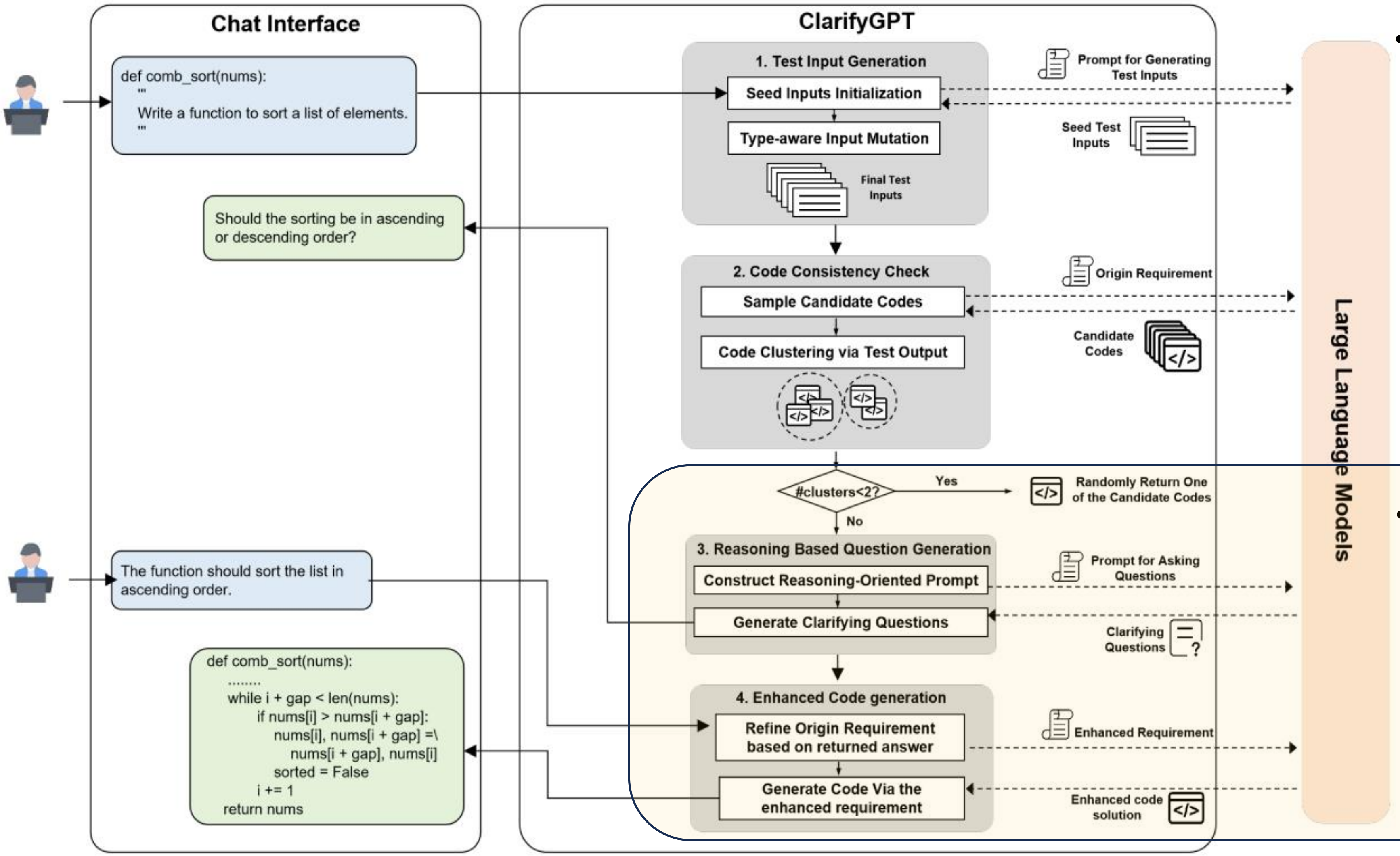
Empowering LLM-based Code Generation with Intention Clarification



Overview of ClarifyGPT

- **Challenge1:** **When to ask clarifying questions?**
  - ➢ **Test Input Generation:** Generate a large number of tests via mutation based on the reqs
  - ➢ **Code Consistency Check:** Run the generated code on the test inputs. If the test outputs are consistent, we consider the requirements are unambiguous

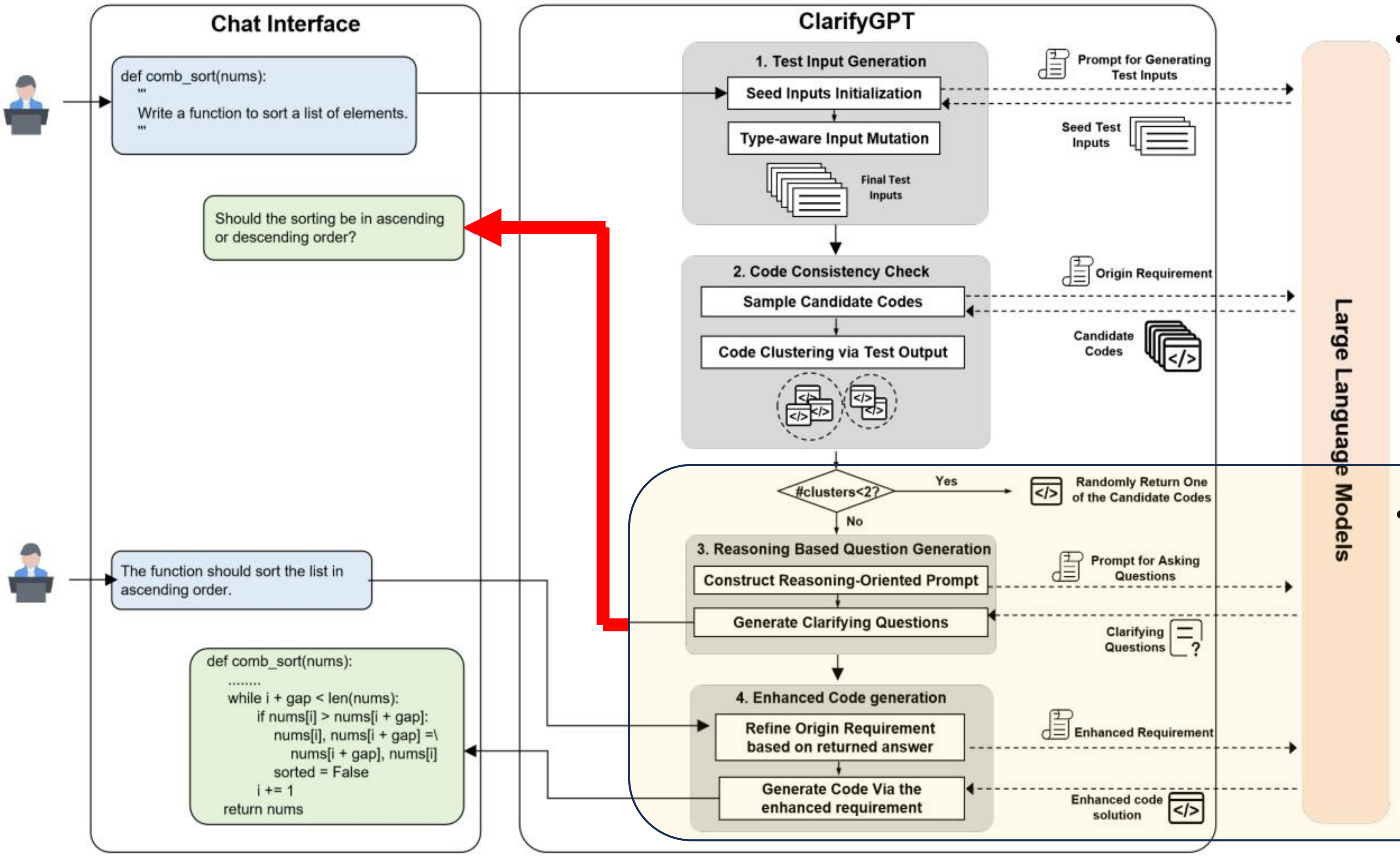
Empowering LLM-based Code Generation with Intention Clarification



Overview of ClarifyGPT

- **Challenge1: When to ask clarifying questions?**
  - **Test Input Generation:** Generate a large number of tests via mutation based on the reqs
  - **Code Consistency Check:** Run the generated code on the test inputs. If the test outputs are consistent, we consider the requirements are unambiguous
- **Challenge2: What questions?**
  - **Reasoning-based Question Generation:** Use LLMs to analyze the reasons for ambiguity and propose targeted questions based on the identified causes.

Empowering LLM-based Code Generation with Intention Clarification

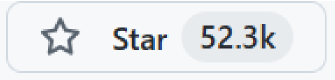


Overview of ClarifyGPT

- **Challenge1: When to ask clarifying questions?**
  - ➢ **Test Input Generation:** Generate a large number of tests via mutation based on the reqs
  - ➢ **Code Consistency Check:** Run the generated code on the test inputs. If the test outputs are consistent, we consider the requirements are unambiguous
- **Challenge2: What questions?**
  - ➢ **Reasoning-based Question Generation:** Use LLMs to analyze the reasons for ambiguity and propose targeted questions based on the identified causes.

ClarifyGPT improves the performance of LLM-based code generation by around 15%

- **Participants**:
  - students, researchers, and developers
  - >3 years of experience with Python
- **Two baselines:**
  - Chain-of-thought
  - GPT-Engineering



- **Datasets:**
  - MBPP-sanitized/ET (427)
  - HumanEval (164)

The Pass@1(%) of ClarifyGPT

Methods	GPT-4		
	MBPP-sanitized	MBPP-ET	Average
Default	70.96	51.52	61.24
CoT	72.68	53.79	63.24
GPT-Engineer	73.77	54.96	64.37
ClarifyGPT (Human Feedback)	80.80	60.19	70.50
Relative Improvement	13.87% ↑	16.83% ↑	15.35% ↑

ClarifyGPT elevates the performance (Pass@1) of GPT-4 on MBPP-sanitized from 70.96% to 80.8%; and elevates its performance on MBPP-ET from 51.52% to 60.19%. The relative improvement is 15.35% on average, outperforming the baselines.

ClarifyGPT with Simulated User Feedback

	Methods	HumanEval	HumanEval-ET	MBPP-sanitized	MBPP-ET	Average
ChatGPT	Default	64.63	57.32	65.57	46.68	58.55
	CoT	68.70	60.37	66.59	49.18	61.21
	GPT-Engineer	66.26	59.76	69.09	50.20	61.33
	ClarifyGPT (Simulated Feedback)	**74.39**	**64.84**	**74.08**	**55.58**	**67.22**
	Relative Improvement	15.10% ↑	13.12% ↑	12.98% ↑	19.07% ↑	15.07% ↑
GPT-4	Default	78.86	70.73	70.96	51.52	68.02
	CoT	80.10	72.56	72.68	53.79	69.78
	GPT-Engineer	79.27	71.75	73.77	54.96	69.94
	ClarifyGPT (Human Feedback)	\	\	**80.80**	**60.19**	**70.50**
	ClarifyGPT (Simulated Feedback)	**87.80**	**78.05**	**78.69**	**58.47**	**75.75**
	Relative Improvement	11.34% ↑	10.35% ↑	10.89% ↑	13.49% ↑	11.52% ↑

ClarifyGPT vs. GPT-Engineering

Case I (Problem 71 in MBPP-sanitized):

```python
def comb_sort(nums):
    ''' Write a function to sort a list of elements. '''
```

Default (ChatGPT)

```python
def comb_sort(nums):
    gap = len(nums)
    shrink_factor = 1.3
    sorted = False
    while not sorted:
        gap = int(gap / shrink_factor)
        if gap <= 1:
            gap = 1
            sorted = True
        i = 0
        while i + gap < len(nums):
            if nums[i] < nums[i + gap]:
                nums[i], nums[i + gap] =\
                    nums[i + gap], nums[i]
                sorted = False
            i += 1
    return nums
```

GPT-Engineer (ChatGPT)

Questions:
1.What should be the input of the sorting function?
2.Should the sorting be in ascending or descending order?
3.Are there any specific constraints for the sorting algorithm?
4.Do you want the entire implementation of this function?
5.Are there any preferred programming language?

Answers:
1.The function should take a list of numbers as input.
2.The function should sort the list in ascending order.
3.The sorting algorithm should be comb sort.
4.Yes, I want the entrie implementation.
5.Please use Python to implement this funtion.

Code Solution:

```python
def comb_sort(nums):
    ........
    while i + gap < len(nums):
        if nums[i] < nums[i + gap]:
            nums[i], nums[i + gap] =\
                nums[i + gap], nums[i]
            sorted = False
        i += 1
    return nums
```

ClarifyGPT (ChatGPT)

Questions:
1. Should the sorting be in ascending or descending order?

Ask targeted question

Answers:
1. The function should sort the list in ascending order.

Code Solution:

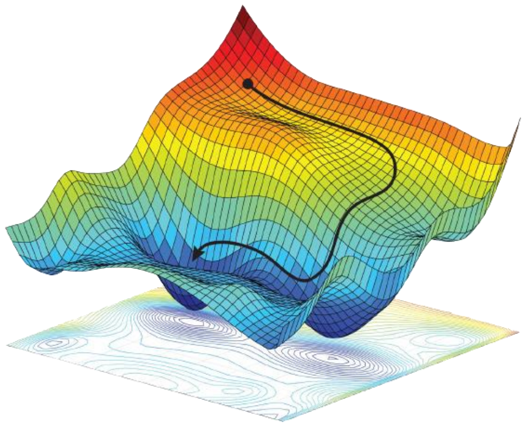
```python
def comb_sort(nums):
    ........
    while i + gap < len(nums):
        if nums[i] < nums[i + gap]:
            nums[i], nums[i + gap] =\
                nums[i + gap], nums[i]
            sorted = False
        i += 1
    return nums
```

- **Pitfalls of GPT-Engineering:**
  - ask questions for every problem (on average 3 more than ClarifyGPT)
  - ask unnecessary questions

Enhancing prompts

EPiC: Search-based Prompt Optimization for LLM-based Code Generation

- **Prompt Optimization**
  - Utilize **search algorithms** to explore variations of prompts to identify those that yield the best responses
  - Better alignment with LLMs' training data, and stimulate better response with certain structures/words/phrases



https://arxiv.org/pdf/2408.11198

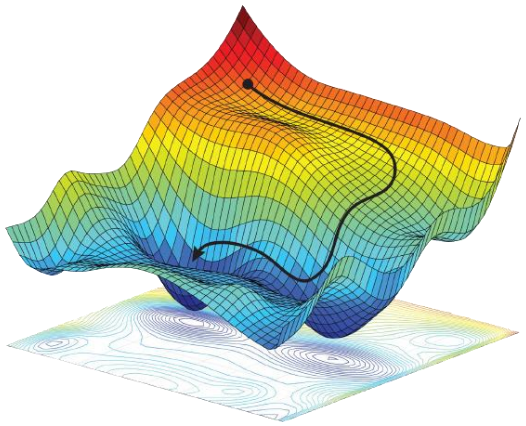
EPiC: Cost-effective Search-based Prompt Engineering of LLMs for Code Generation

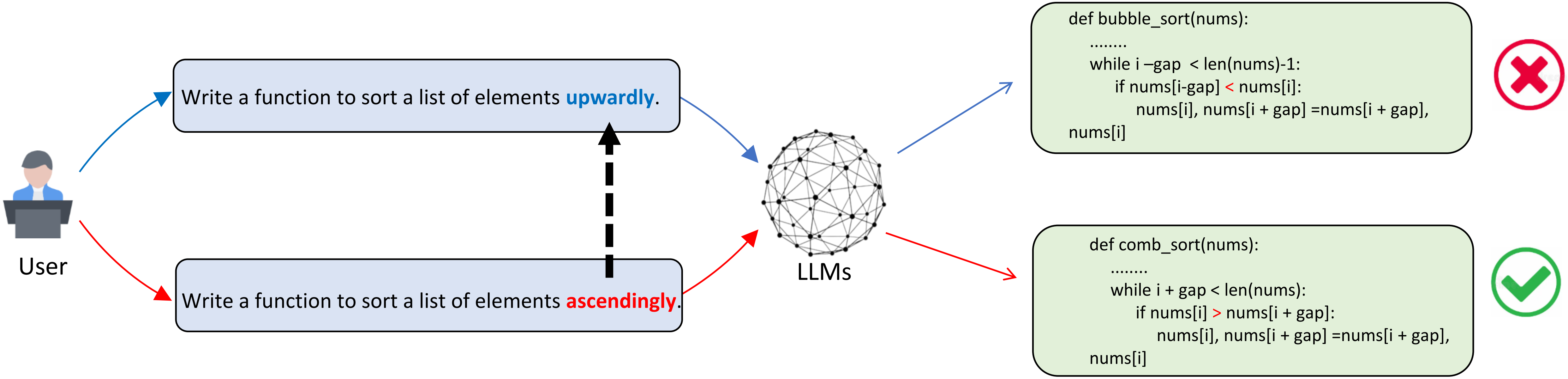
H Taherkhani, M Sepindband, HV Pham, S Wang, H Hemmati

Enhancing prompts

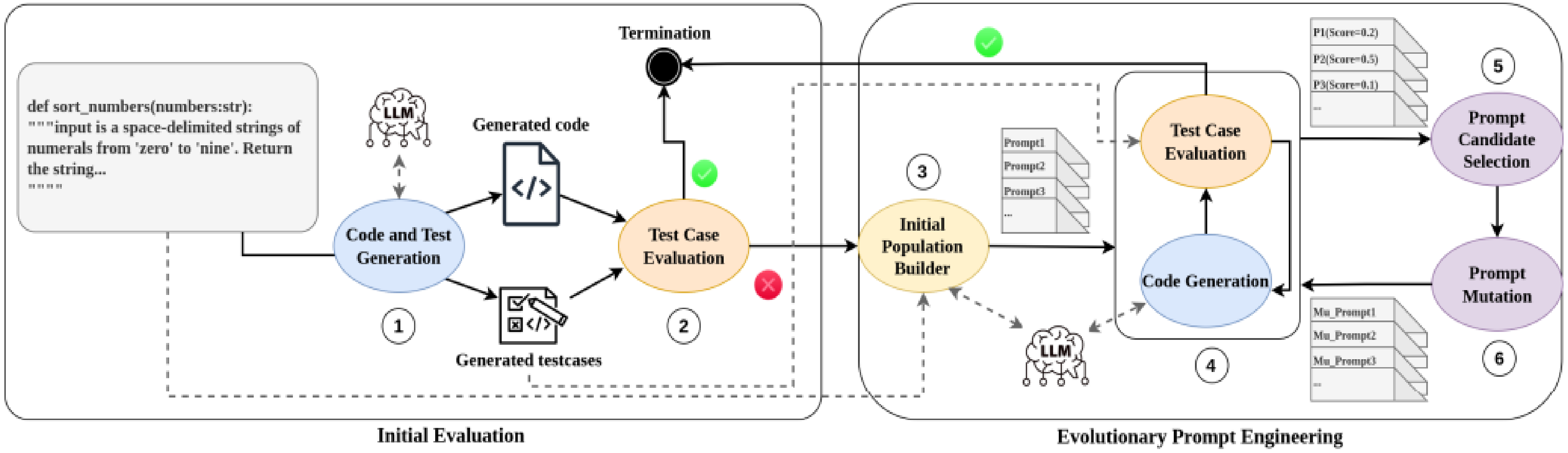
EPiC: Search-based Prompt Optimization for LLM-based Code Generation

- **Prompt Optimization**
  - Utilize **search algorithms** to explore variations of prompts to identify those that yield the best responses
  - Better alignment with LLMs' training data, and stimulate better response with certain structures/words/phrases





Overview of EPiC



- **fitness function**
  - pass rate of tests
- **mutation approaches**
  - LLM-based
  - similar_words_replace
- **population**
  - N * 10

1. **Generate initial tests and solution**
2. **Evaluate the generated code**
3. **Build initial population with LLM**
4. **Evaluate each prompt and calculate the fitness score**
5. **Select the candidate prompts for mutation**
6. **Mutate prompts and re-generating solutions**

EPiC outperforms SOTA in pass@1, $cost, and time

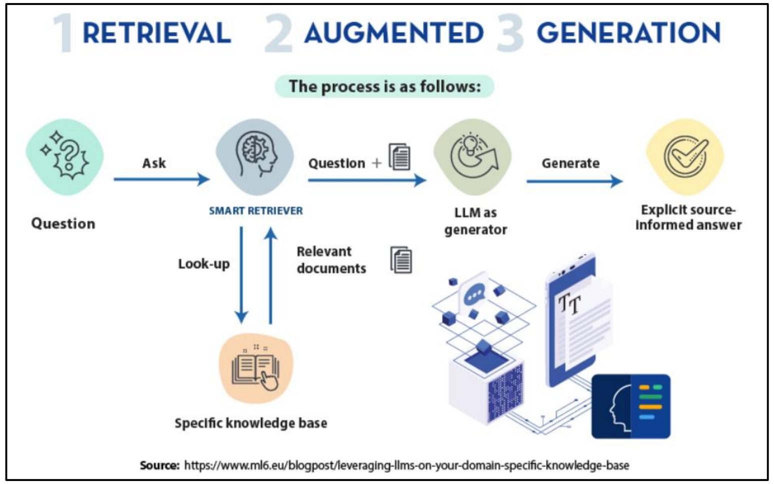
- **Three baselines:**
  - Reflexion (verbal feedback +RL)
  - LDB (feedback + COT)
  - LATS (agent + search)
- **Tow datasets**:
  - HumanEval
  - MBPP

Dataset	Tool	pass@1	cost	time(mins)
Humaneval	Reflexion	%87	2	37
Humaneval	LDB	%92	$3.2	43
Humaneval	LATS	%91	16.51	151
Humaneval	EPiC	%94	$3.5	46
MBPP	Reflexion	%71	4.91	68
MBPP	LDB	%73	$13.81	103
MBPP	LATS	%76	80.2	1403
MBPP	EPiC	%79	$16.4	210

EPiC outperforms the SOTA baselines by %1 to %3 on HumanEval and %2 to %7 on MBPP with costs that are either lower or comparable to prior studies.

The Impact of Different Knowledge Base Sources on RAG-based Unit Test Generation

- **Retrieval-Augmented Generation (RAG)**
  - Allows LLMs access to the latest and more domain-specific information
  - Helps ground the output in factual information, reducing the likelihood of hallucinations



https://arxiv.org/abs/2310.10508

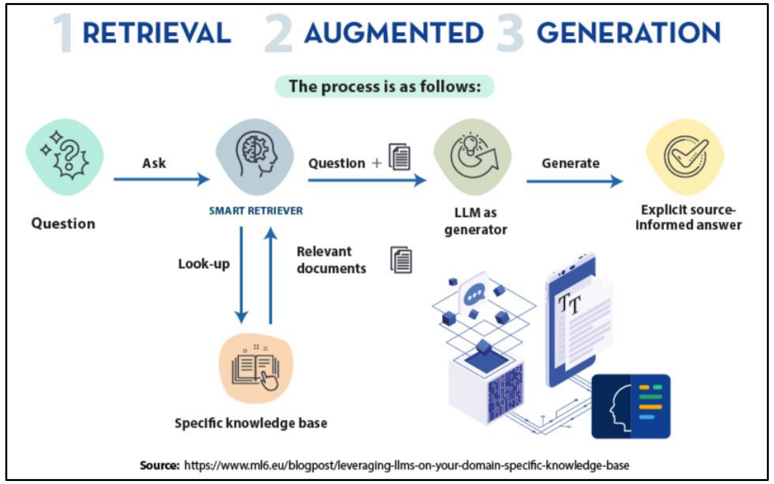
Prompt Engineering or Fine Tuning: An Empirical Assessment of Large Language Models in Automated Software Engineering Tasks

J Shin, C Tang, T Mohati, M Nayebi, S Wang, H Hemmati

Enhancing prompts

The Impact of Different Knowledge Base Sources on RAG-based Unit Test Generation

- **Retrieval-Augmented Generation (RAG)**
  - ➢ Allows LLMs access to the latest and more domain-specific information
  - ➢ Helps ground the output in factual information, reducing the likelihood of hallucinations



- **External Sources for RAG-based test generation:**

GitHub

Code Repo

stack overflow

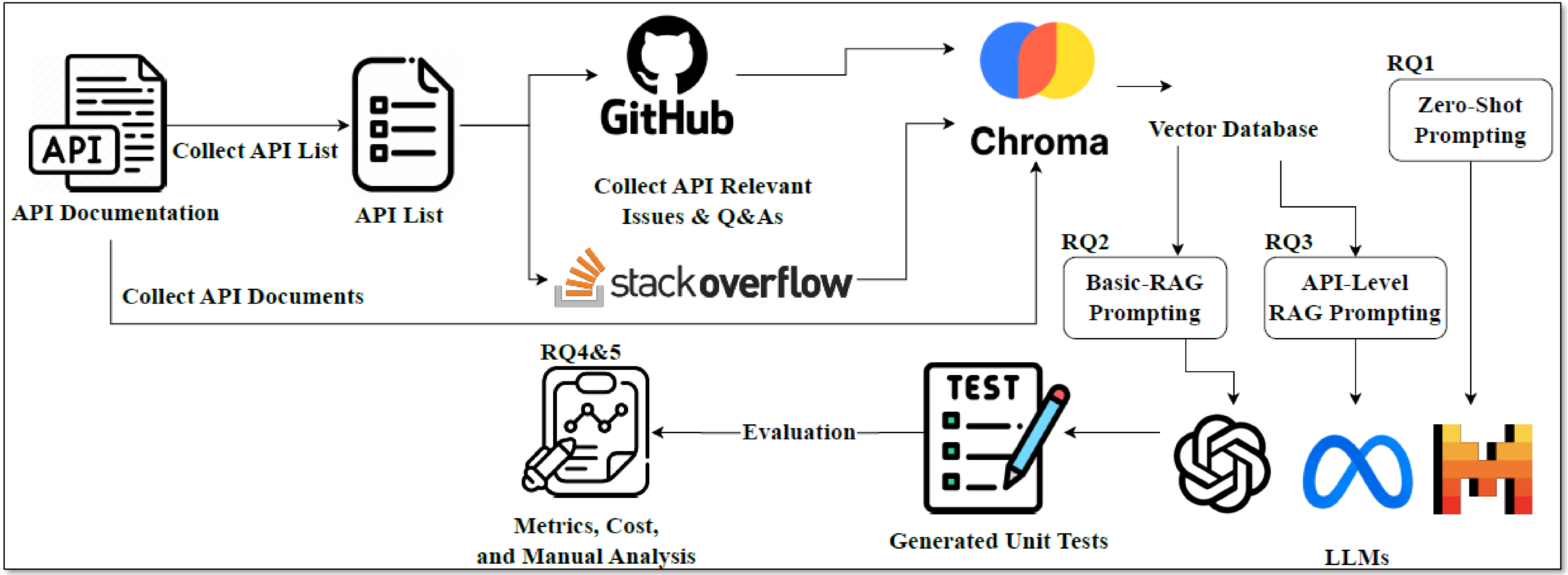
Q&A Knowledge

API

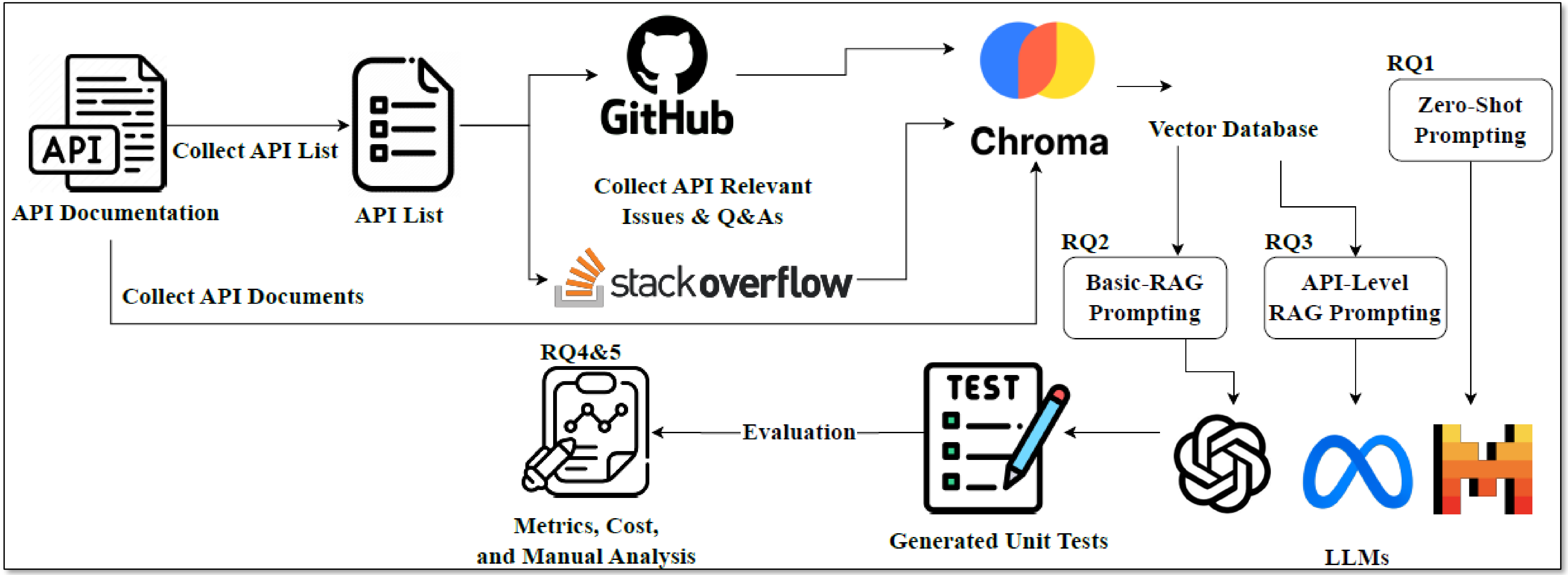
Documents

.....

Impact of different external resources of RAG-based test generation

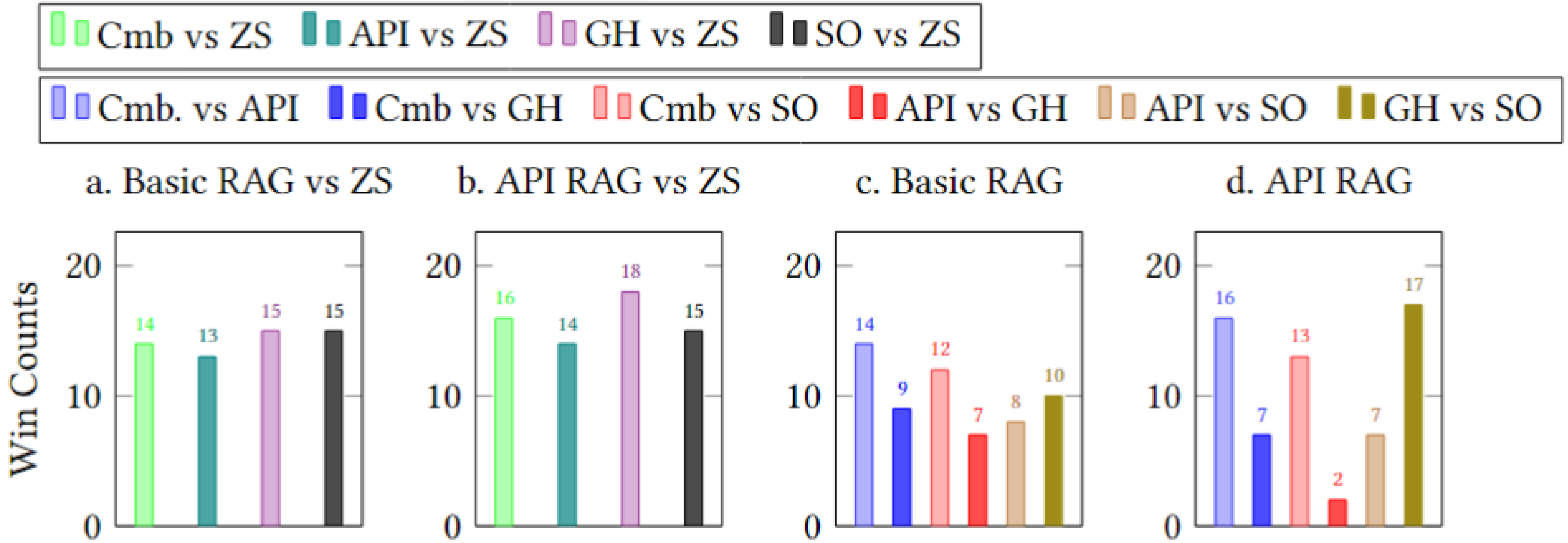


Impact of different external resources of RAG-based test generation



- **Four baselines:**
  - GPT-3.5-turbo
  - GPT-4o
  - Mistral 8x22B instruct
  - Llama 3.1 405B instruct
- **Five DL infrastructure libs:**
  - TensorFlow
  - PyTorch
  - Sk-learn
  - Google Jax
  - XGBoost
- **Four metrics:**
  - Parse rate
  - Execution rate
  - Pass rate
  - Code Coverage
- **Two RAGs:**
  - Basic
  - API

Performance of RAGs with different settings



Win counts (based on code coverage) of the RAG approaches vs the zero-shots (**ZS**). **Cmb** denotes combined RAG, **API** denotes API documents, **GH** denotes GitHub issues, and **SO** denotes StackOverflow Q&As.

1. RAG could **improve the code coverage not the syntactical correctness** of unit test cases.
2. API-level RAG generally performs better than Basic (project-level) RAG.
3. **GitHub issues benefit RAG** the most among the three examined sources.

Benefits of GitHub issues for RAGs: covering more corner cases

```
# Document that led to the generation
...
manager = tf.train.CheckpointManager(checkpoint=checkpoint, directory='./delete_me/', max_to_keep=2)
for i in range(1000):
    manager.save(i)
manager2 = tf.train.CheckpointManager(checkpoint=checkpoint, directory='./delete_me/', max_to_keep=2)
for i in range(1000, 2000):
    manager2.save(i)
...
# Generated Unit Test Case
def test_multiple_checkpoint_managers(self):
    # Test with multiple checkpoint managers
    manager1 = tf.train.CheckpointManager(self.checkpoint, self.checkpoint_dir, max_to_keep=2)
    for i in range(2):
        manager1.save()

    manager2 = tf.train.CheckpointManager(self.checkpoint, self.checkpoint_dir, max_to_keep=2)
    for i in range(2, 4):
        manager2.save()
...
# Unique Covered Line from Unit Under Test
...
def __init__(self, checkpoint, directory, max_to_keep, ... ):
...
  self._latest_checkpoint = recovered_state.model_checkpoint_path           #UNIQUE#
  self._last_preserved_timestamp = recovered_state.last_preserved_timestamp #UNIQUE#
  if current_clock < self._last_preserved_timestamp:                        #UNIQUE#
    ...
  all_timestamps = recovered_state.all_model_checkpoint_timestamps          #UNIQUE#
  all_paths = recovered_state.all_model_checkpoint_paths                    #UNIQUE#
  del recovered_state  # Uses modified values from now on                   #UNIQUE#
  if not all_timestamps:                                                    #UNIQUE#
    all_timestamps = [self._last_preserved_timestamp] * len(all_paths)

  for filename, timestamp in zip(all_paths, all_timestamps):                #UNIQUE#
    timestamp = min(timestamp, current_clock)                               #UNIQUE#
    if timestamp > self._last_preserved_timestamp:                          #UNIQUE#
      self._maybe_delete[filename] = timestamp                              #UNIQUE#

...
```

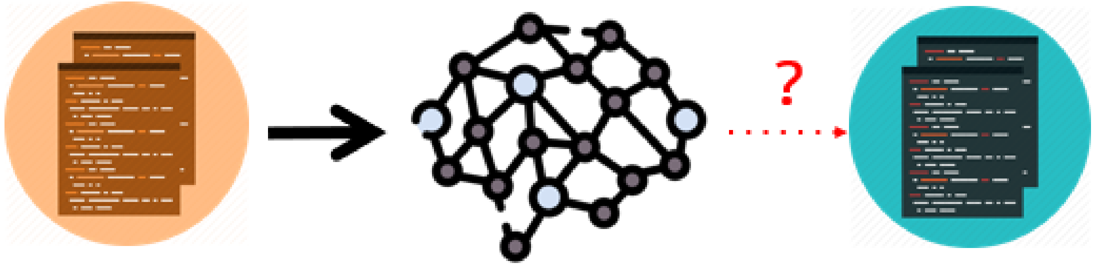
GitHub issues provide unique knowledge

- **Structured and Context-Rich Information:**
  - logs, stack traces, code snippets
- **Detailed Problem Context:**
  - exact inputs used and the method calls that led to the problem

Domain Adaptation for Code Model-based Unit Test Case Generation

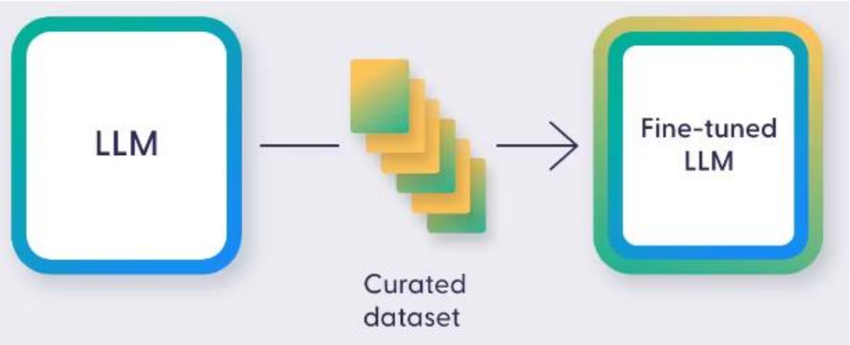
- **Domain shift:**

occurs when a machine learning model is trained on data from one domain but is later applied to data from a different domain, leading to a drop in performance.



- **Fine-tuning:**

Involves continuing training with datasets from a specific **task** and adjusting the model weights.

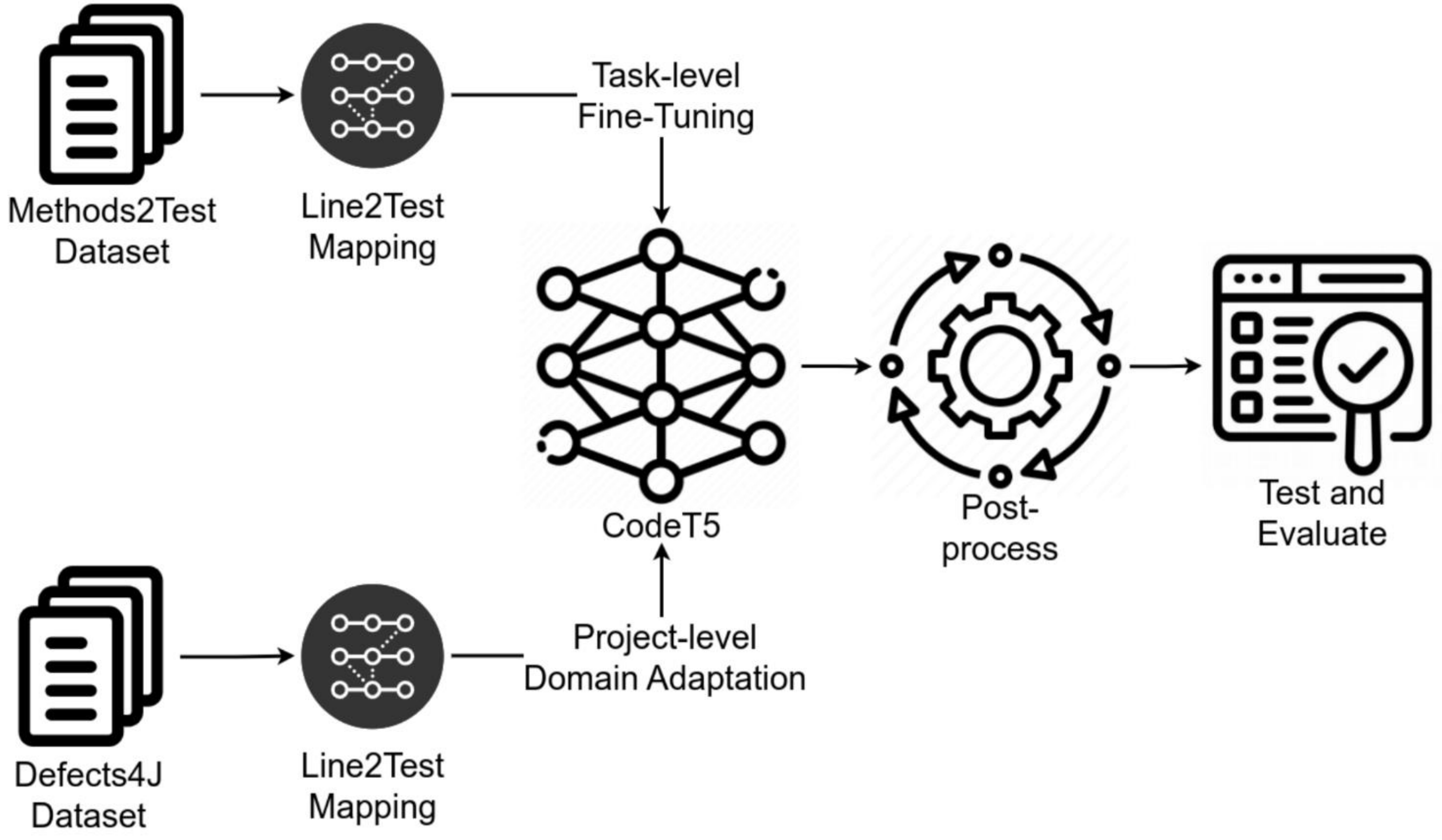


- **Domain Adaptation**

Fine-grained fine-tuning with datasets from a specific domain or **project**.

Domain Adaptation for Code Model-Based Unit Test Case Generation
J Shin, S Hashtroudi, H Hemmati, S Wang
ISSTA 2024

Our Approach: Fine-tuning + Domain adaptation



1. Test Mapping

- Which lines are covered by which tests

2. Fine-tune on "task-level"

- *Methods2Test* data (780K)

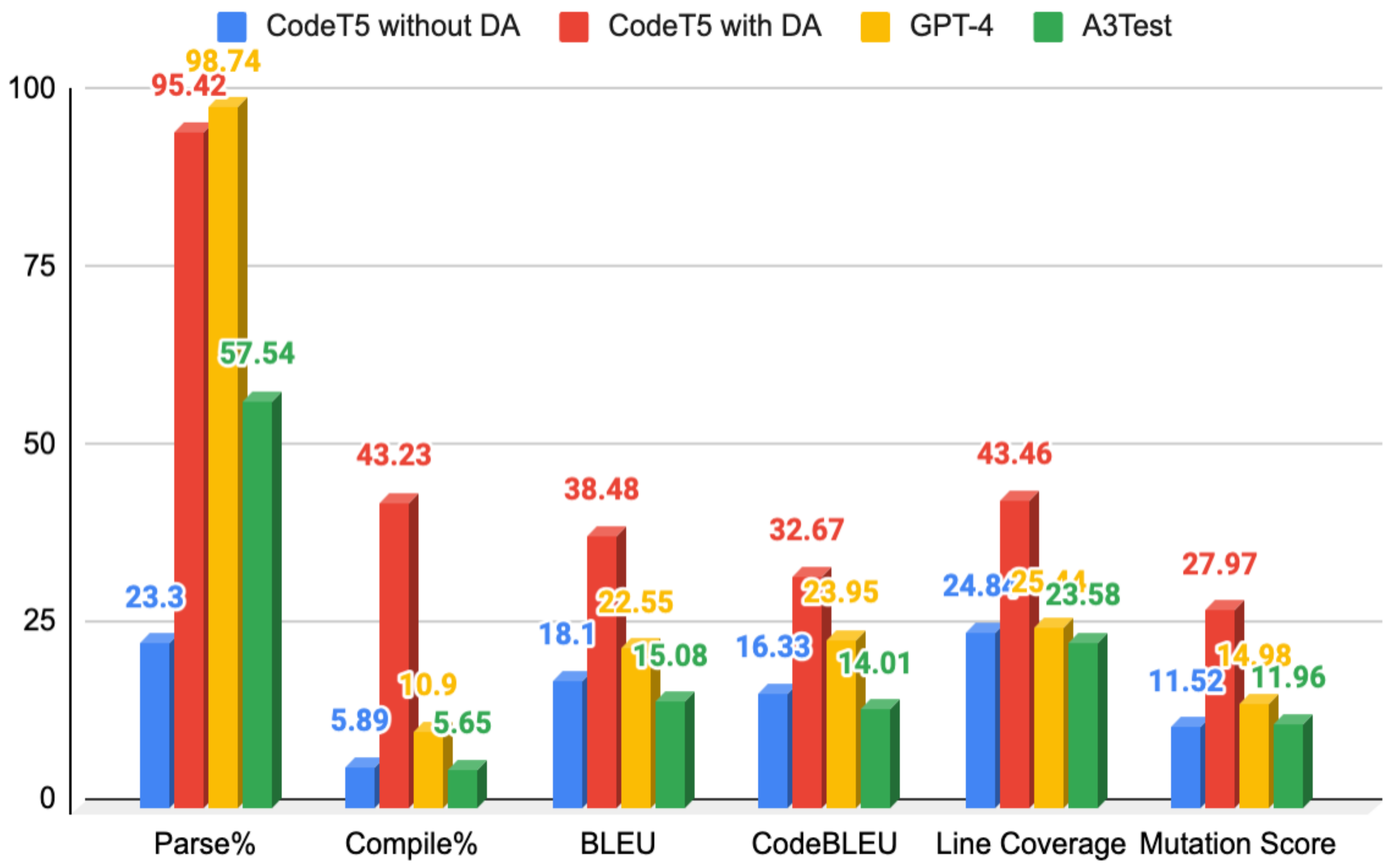
3. DA on "project-level"

- *Defects4J* (20% code from projects)

4. Post-processing

- AST parsability check
- Remove existing tests
- Inject unit tests

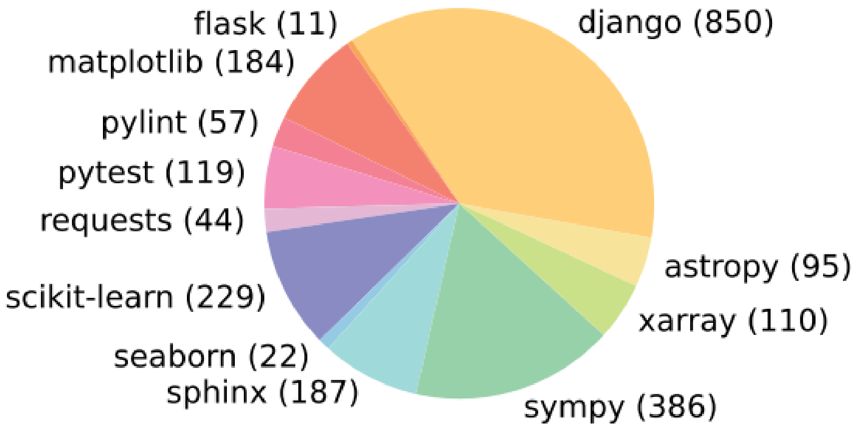
CodeT5 with and without DA vs. GPT-4 and A3Test



- **Base model:**
  - CodeT5 (220M parameters)
- **Two baselines:**
  - GPT-4
  - A3Test (PLBart)
- **Five metrics:**
  - Parse rate
  - Execution rate
  - BLEU/CodeBLEU
  - Line Coverage
  - Mutation Score

Benchmark Reliability

SWE-Bench+: Enhanced Coding Benchmark for LLMs



- A Groundbreaking evaluation framework designed to assess the capabilities of LLMs in resolving GitHub Issues.
- Address the limitations of traditional benchmarks that are synthetic or simplified (they are complex and real).
- They include issues with test cases expecting the LLMs to pass them.

Leaderboard

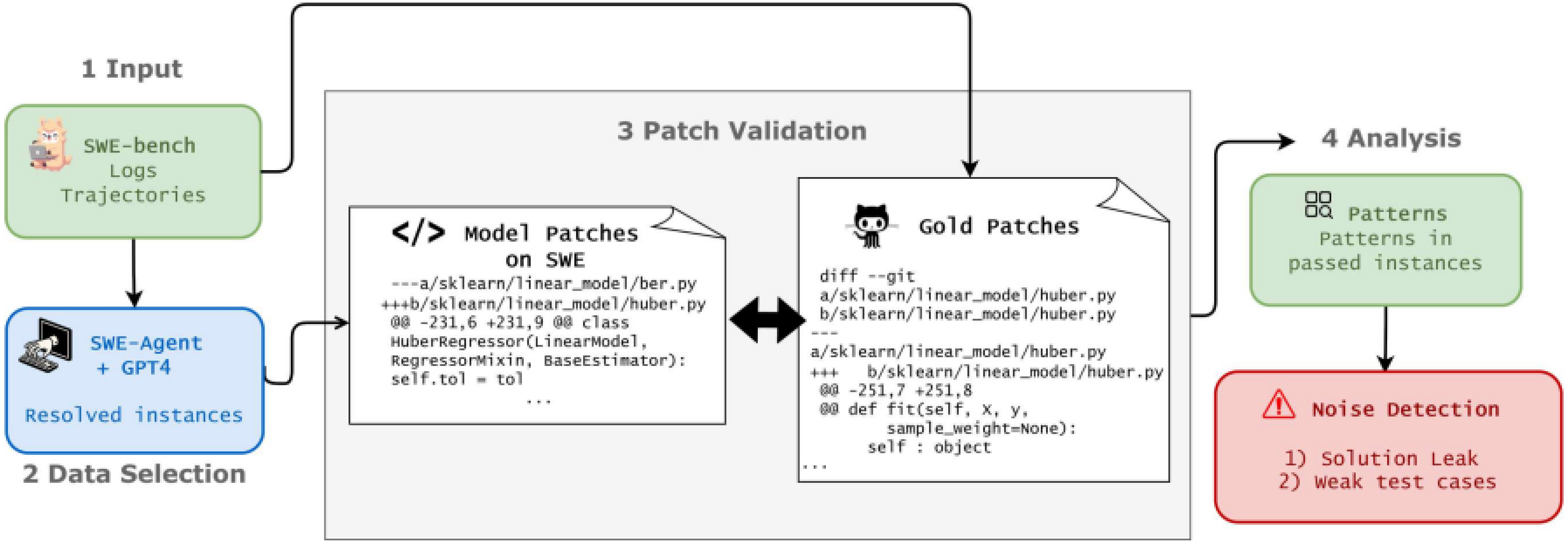
Lite | Verified | Full

Model	% Resolved	Org	Date	Logs	Trajs	Site
🥇 Honeycomb	22.06		2024-08-20	✓	✓	🔗
🥈 Amazon Q Developer Agent (v20240719-dev)	19.75	aws	2024-07-21	✓	✓	🔗
🥉 Factory Code Droid	19.27		2024-06-17	✓	-	🔗
AutoCodeRover (v20240620) + GPT 4o (2024-05-13)	18.83		2024-06-28	✓	-	🔗
🤠 ✅ SWE-agent + Claude 3.5 Sonnet	18.13		2024-06-20	✓	✓	-

arXiv preprint arXiv:2410.06992
SWE-Bench+: Enhanced Coding Benchmark for LLMs
R Aleithan, H Xue, MM Mohajer, E Nnorom, G Uddin, S Wang

Robustness Analysis of SWE-Bench

are the LLMs actually resolving the issues in the SWE-bench?



Overview of our manual analysis

- **Model:**
  - ➢ SWE-Agent + GPT4
- **Data:**
  - ➢ 2k raw issues (titles, tests, gold patches)
  - ➢ **251 successfully fixed issues**
    - ➢ generated patches
    - ➢ tests
- **Patch Validation Study:**
  - ➢ each patch v.s gold patch
  - ➢ review logs, issue descriptions, tests

64% of the solved issues are suspicious

Pattern	Numbers (percentage)	Root cause
Solution leak	82 (32.67%)	solution leakage
Incorrect fixes	32 (12.75%)	weak tests
Different files/functions changed	9 (3.59%)	weak tests
Incomplete fixes	37 (14.74%)	weak tests

64% of the solved issues are suspicious

Issue Report - Comments

ruoyu0088 commented on Apr 17, 2019 • edited

I use `lambdify()` to generate some functions and save the code for further use. But the generated code for `Indexed` operation has some warnings which can be confirmed by following code;

```
from sympy import *
p = IndexedBase("p")

pycode(p[0])
```

the output is

```
  # Not supported in Python:
  # Indexed
p[0]
```

We should add following method to `PythonCodePrinter`:

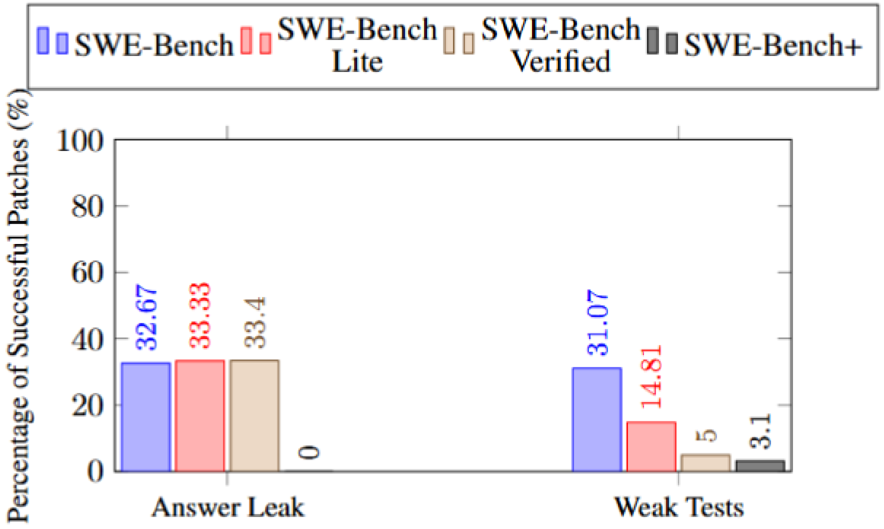
```
def _print_Indexed(self, expr):
    base, *index = expr.args
    return "{}[{}]".format(str(base), ", ".join([self._print(ind) for ind in index]))
```

Generated Patch

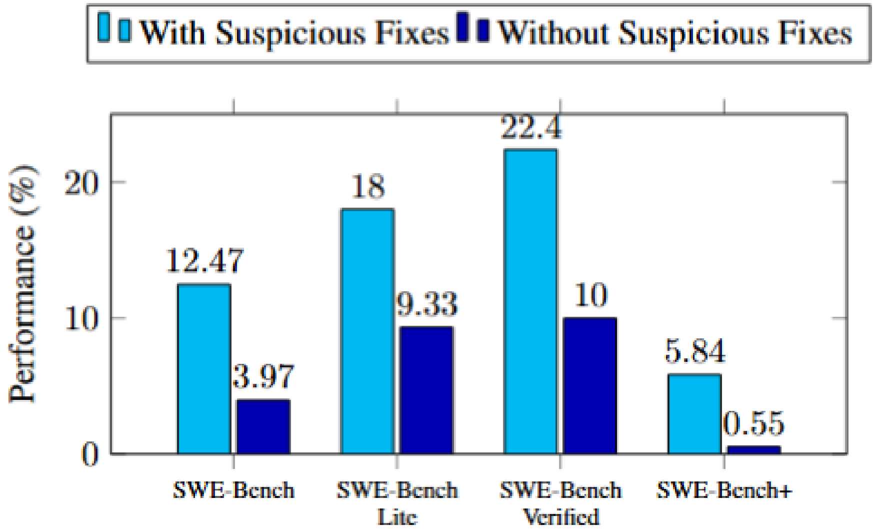
```
sympy/printing/pycode.py
        PREC = precedence(expr)
        return self._operators['not'] + self.parenthesize(expr.args[0], PREC)

+    def _print_Indexed(self, expr):
+        base, *index = expr.args
+        return "{}[{}]".format(str(base), ", ".join([self._print(ind) for ind in index]))
```

LLMs' performance on clean data drops by 90%

Pattern	Numbers (percentage)	Root cause
Solution leak	82 (32.67%)	solution leakage
Incorrect fixes	32 (12.75%)	weak tests
Different files/functions changed	9 (3.59%)	weak tests
Incomplete fixes	37 (14.74%)	weak tests



(a) Answer Leak vs Weak Tests across Datasets



(b) Performance of SWE-Agent + GPT-4 across datasets.

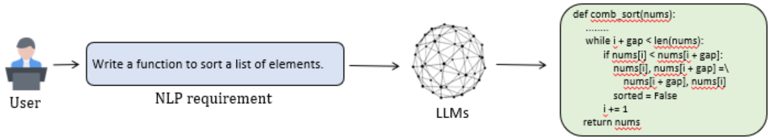
From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

Enhancing prompts

ClarifyGPT: Empowering LLM-based Code Generation with Intention Clarification (FSE 2024)

- **Auto Code Generation**
  - ➢ Converting user-provided natural language requirements to executable code
  - ➢ Improving software development efficiency

https://survey.stackoverflow.co/2023/#ai



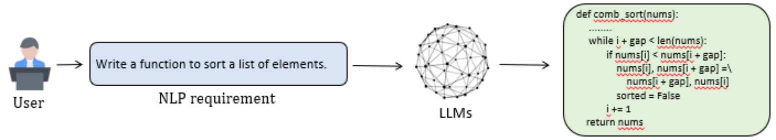
From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

Enhancing prompts

ClarifyGPT: Empowering LLM-based Code Generation with Intention Clarification (FSE 2024)

- **Auto Code Generation**
  - ➢ Converting user-provided natural language requirements to executable code
  - ➢ Improving software development efficiency

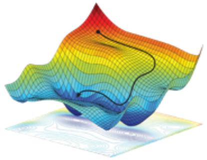
https://survey.stackoverflow.co/2023/#ai

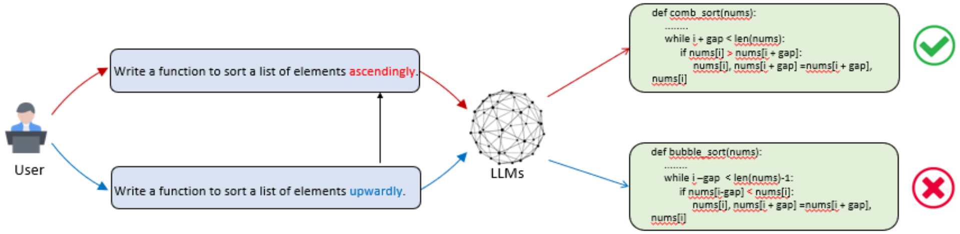


Enhancing prompts

EPiC: Search-based Prompt Optimization for LLM-based Code Generation (TSE under review)

- **Prompt Optimization**
  - ➢ Utilize **search algorithms** to explore variations of prompts to identify those that yield the best responses
  - ➢ Better alignment with LLMs' training data, and stimulate better response with certain structures/words/phrases





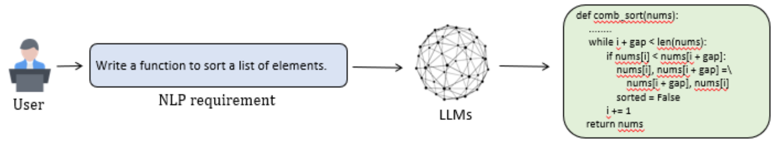
From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

Enhancing prompts

ClarifyGPT: Empowering LLM-based Code Generation with Intention Clarification (FSE 2024)

- **Auto Code Generation**
  - ➢ Converting user-provided natural language requirements to executable code
  - ➢ Improving software development efficiency

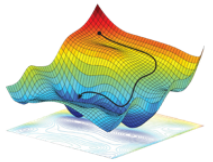
https://survey.stackoverflow.co/2023/#ai

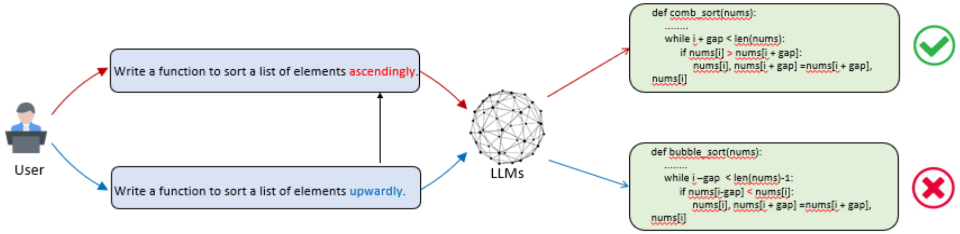


Enhancing prompts

EPiC: Search-based Prompt Optimization for LLM-based Code Generation (TSE under review)

- **Prompt Optimization**
  - ➢ Utilize **search algorithms** to explore variations of prompts to identify those that yield the best responses
  - ➢ Better alignment with LLMs' training data, and stimulate better response with certain structures/words/phrases

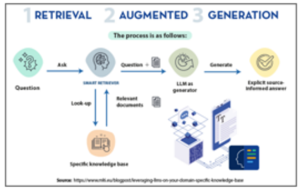




Enhancing prompts

The Impact of Different Knowledge Base Sources on RAG-based Unit Test Generation (under review)

- **Retrieval-Augmented Generation (RAG)**
  - ➢ Allows LLMs access to the latest and more domain-specific information
  - ➢ Helps ground the output in factual information, reducing the likelihood of hallucinations



- **External Sources for RAG**

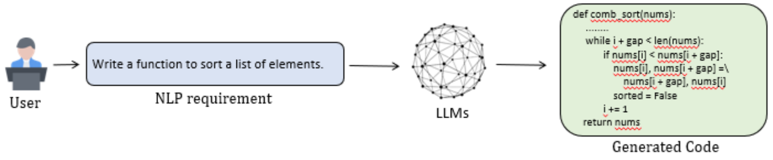
GitHub — Code Repo | stack overflow — Q&A Knowledge | API — Documents | .....

From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

Enhancing prompts — ClarifyGPT: Empowering LLM-based Code Generation with Intention Clarification (FSE 2024)

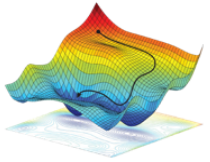
- **Auto Code Generation**
  - ➢ Converting user-provided natural language requirements to executable code
  - ➢ Improving software development efficiency

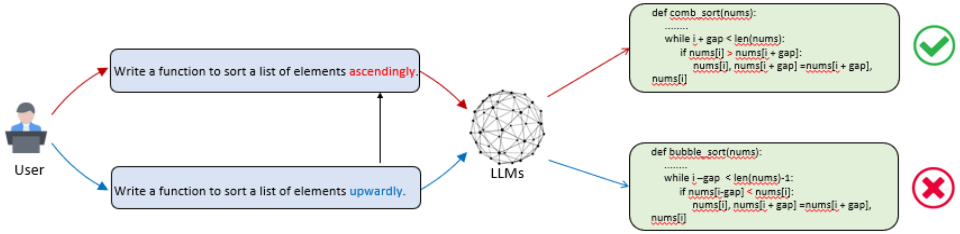
https://survey.stackoverflow.co/2023/#ai



Enhancing prompts — EPiC: Search-based Prompt Optimization for LLM-based Code Generation (TSE under review)

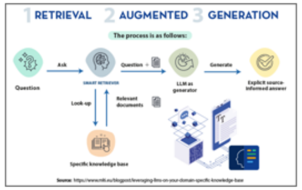
- **Prompt Optimization**
  - ➢ Utilize **search algorithms** to explore variations of prompts to identify those that yield the best responses
  - ➢ Better alignment with LLMs' training data, and stimulate better response with certain structures/words/phrases





Enhancing prompts — The Impact of Different Knowledge Base Sources on RAG-based Unit Test Generation (under review)

- **Retrieval-Augmented Generation (RAG)**
  - ➢ Allows LLMs access to the latest and more domain-specific information
  - ➢ Helps ground the output in factual information, reducing the likelihood of hallucinations



- **External Sources for RAG**

GitHub — **Code Repo**

stack overflow — **Q&A Knowledge**

API — **Documents**

.....

Fine-tuning LLMs — Domain Adaptation for Code Model-based Unit Test Case Generation (ISSTA 2024)

- **Domain shift:** occurs when a machine learning model is trained on data from one domain but is later applied to data from a different domain, leading to a drop in performance.

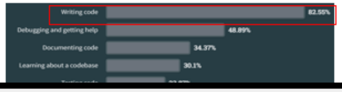
- **Fine-tuning:** involves continuing training with datasets from a specific task and adjusting the model weights.

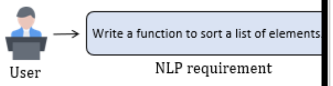
- **Domain Adaptation** fine-grained Fine-tuning with datasets from a specific domain or project.

From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

Enhancing prompts — ClarifyGPT: Empowering LLM-based Code Generation with Intention Clarification (FSE 2024)

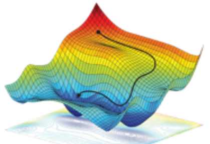
- Auto Code Generation
  - Converting user-provided natural language requirements to executable code
  - Improving software development

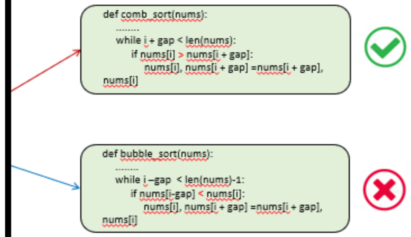




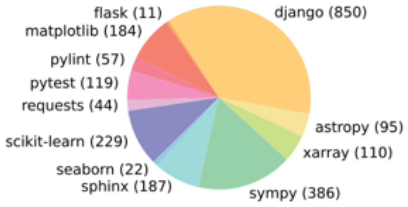
Enhancing prompts — EPiC: Search-based Prompt Optimization for LLM-based Code Generation (TSE under review)

- Prompt Optimization
  - Utilize **search algorithms** to explore variations of





Benchmark Reliability — SWE-Bench+: Enhanced Coding Benchmark for LLMs



- A Groundbreaking evaluation framework designed to assess the capabilities of LLMs in resolving GitHub Issues.
- Address the limitations of traditional benchmarks that are synthetic or simplified (they are complex and real).
- They include issues with test cases expecting the LLMs to pass them.

Leaderboard

Lite | Verified | Full

Model	% Resolved	Org	Date	Logs	Trajs	Site
Honeycomb	22.06		2024-08-20	✓	✓	
Amazon Q Developer Agent (v20240719-dev)	19.75	aws	2024-07-21	✓	✓	
Factory Code Droid	19.27		2024-06-17	✓	-	
AutoCodeRover (v20240620) + GPT 4o (2024-05-13)	18.83		2024-06-28	✓	-	
✅ SWE-agent + Claude 3.5 Sonnet	18.13		2024-06-20	✓	✓	-

arXiv preprint arXiv:2410.06992
SWE-Bench+: Enhanced Coding Benchmark for LLMs
R Aleithan, H Xue, MM Mohaier, E Nnorom, G Uddin, S Wang

Enhancing prompts — The Impact of Dif… RAG-based Unit T…

- Retrieval-Augmented Generati…
  - Allows LLMs access to the latest and more … information
  - Helps ground the output in factual informa… likelihood of hallucinations
- External Sources for RAG

Code Repo | Q&A Knowledge | Documents | .....

…Model-based Unit Test …)

- Fine-tuning: involves continuing training with datasets from a specific task and adjusting the model weights.

- Domain Adaptation: fine-grained Fine-tuning with datasets from a specific domain or project.

From Prompt Enhancement to Fine-Tuning: Improving Automatic SE Tasks with LLMs

https://www.eecs.yorku.ca/~wangsong/

wangsong@yorku.ca

We are hiring (10 faculty positions):

https://lassonde.yorku.ca/about/careers/faculty-recruitment