Large Language Models for Code Intelligence Tasks

LYU2301

Supervisor: Professor Michael R. Lyu

Presenter: Canran Liu, Xingyun Ma

香港中文大學
The Chinese University of Hong Kong

Outline

- Introduction
- Background
- Dataset Processing & Benchmark Creation
- Research Questions
- Experiments
- Conclusion
- Future Work

Introduction - Motivation

- Research around LLMs for coding tasks has emerged as a popular topic.
- LLMs for code have achieved remarkable results in code-related tasks.

ChatGPT

code llama

GitHub Copilot

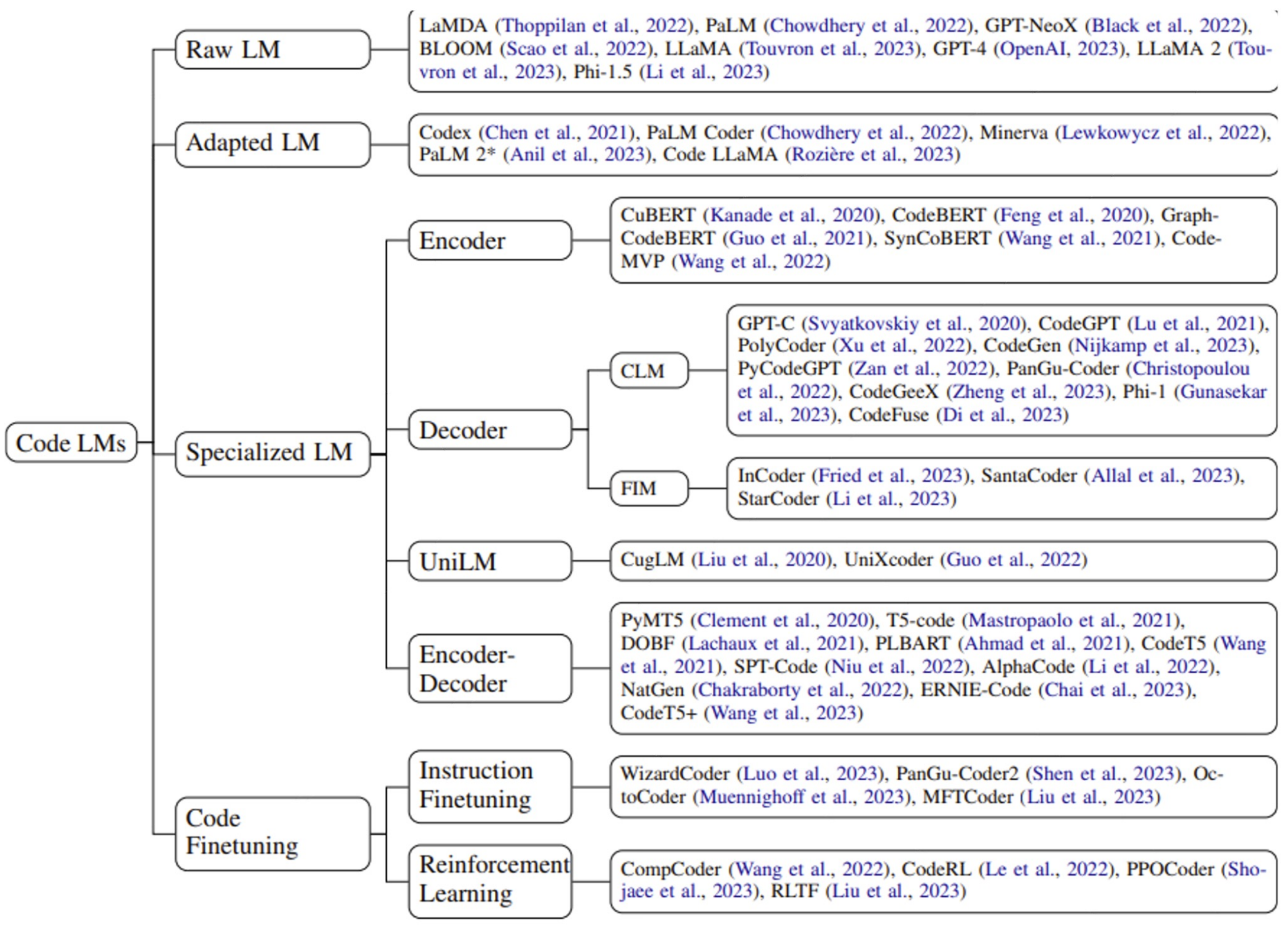
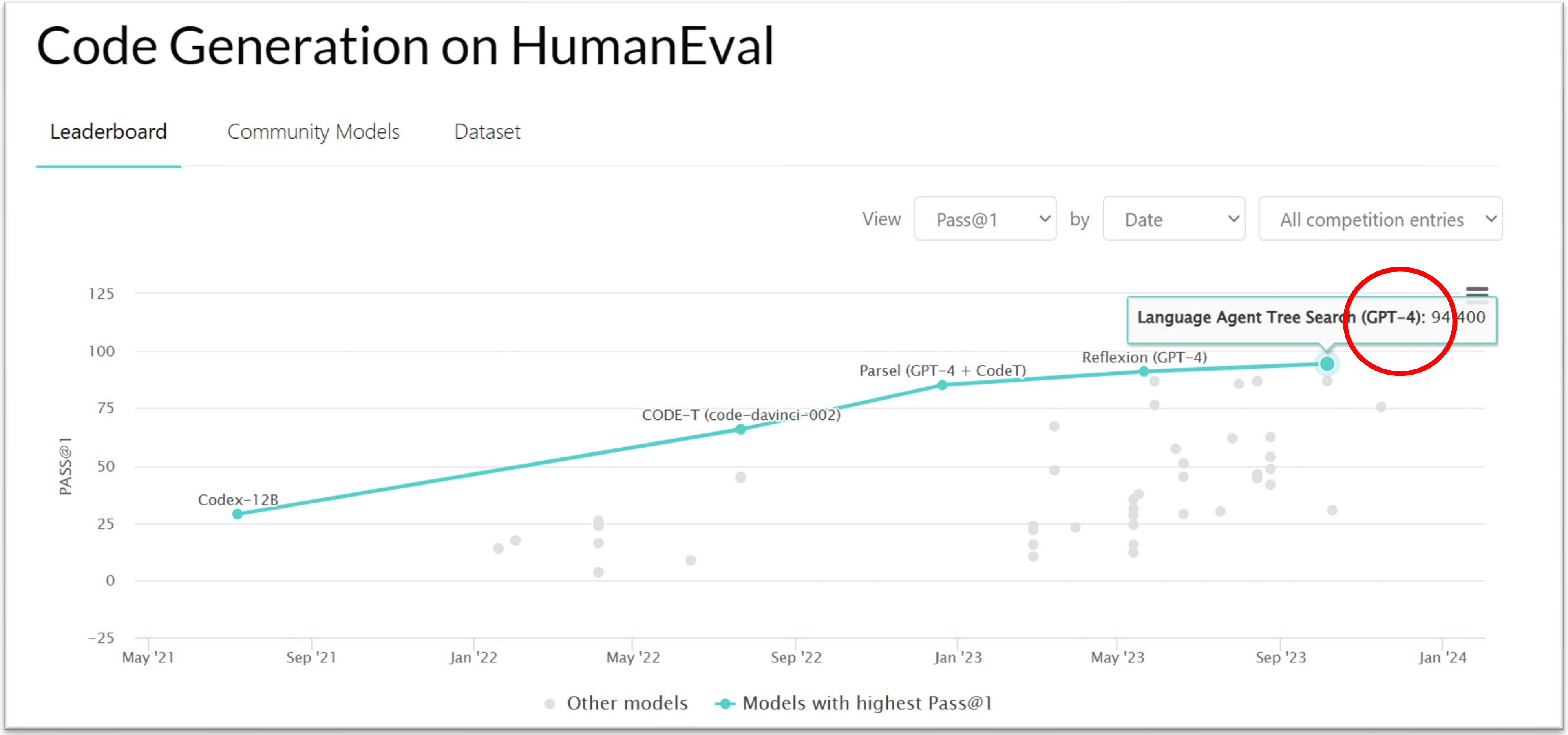


Figure 1: Our taxonomy of pretrained language models for code.

Introduction - Motivation

- Significant progress of research on the accuracy of LLMs in code generation.



Introduction - Motivation

- A simple example to show that there is room for improvement in the efficiency of the generated code

O(nlogn)

Certainly! You can find the kth largest number in a list in Python using a variety of approaches. One common approach is to use the `sorted` function or a heap data structure. Here's an example using the `sorted` function:

python Copy code

```python
def find_kth_largest(nums, k):
    # Sort the list in descending order
    nums.sort(reverse=True)

    # Return the kth largest element
    return nums[k - 1]

# Example usage
my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
k = 3
result = find_kth_largest(my_list, k)
print(f"The {k}th largest number is: {result}")
```

In this code, we first sort the `nums` list in descending order using the `sort` method with `reverse=True`. Then, we simply return the element at index `k - 1`, as Python lists are zero-indexed.

You can replace `my_list` and `k` with your own list and the desired value of `k` to find the kth largest number in your list.

LC Can you improve the time complexity?

O(n)

Certainly! You can improve the time complexity to O(n) on average by using the QuickSelect algorithm. QuickSelect is a modified version of the QuickSort algorithm and is well-suited for finding the kth largest (or smallest) element in an unsorted list. Here's a Python implementation of QuickSelect:

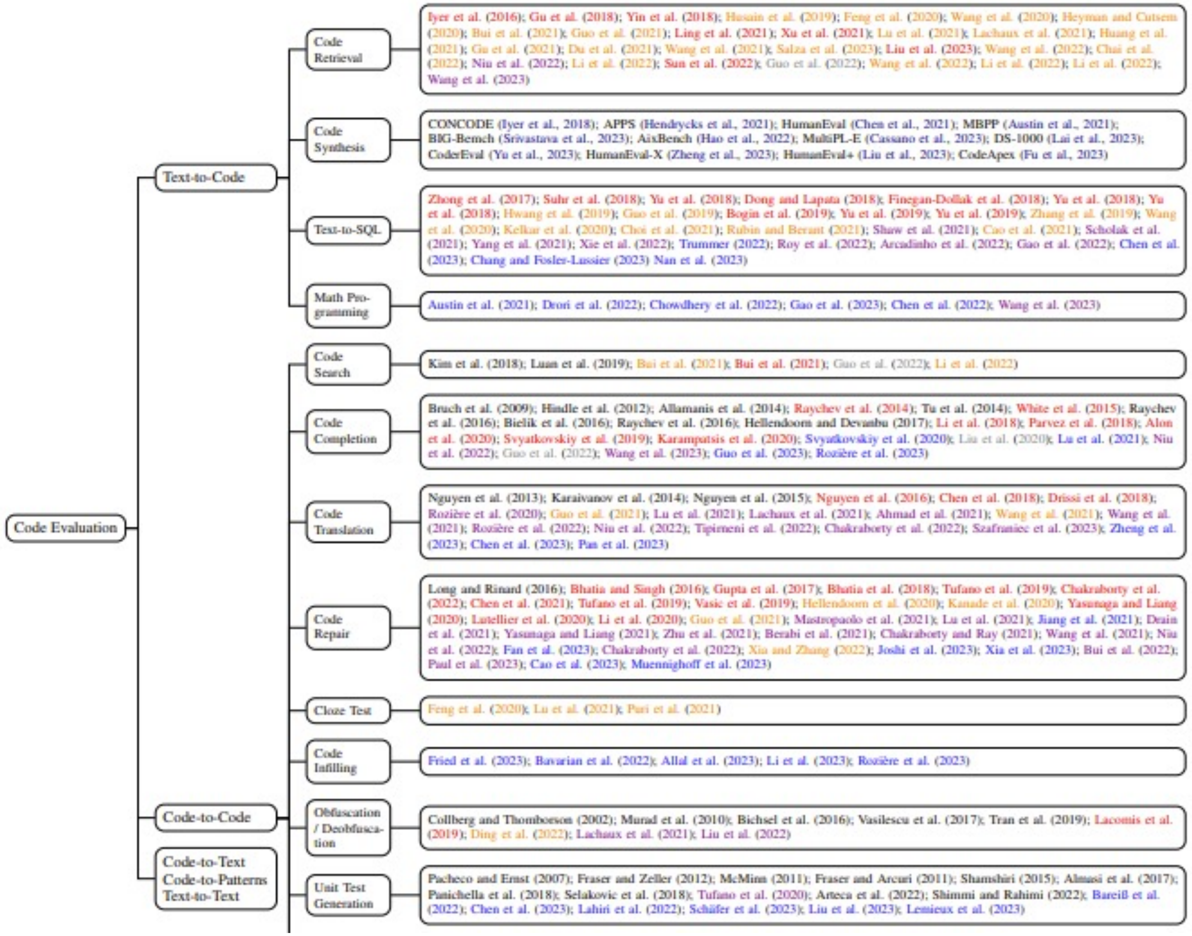
python Copy code

```python
import random

def partition(nums, low, high):
    pivot_index = random.randint(low, high)
    nums[pivot_index], nums[high] = nums[high], nums[pivot_index]
    pivot = nums[high]
    i = low
    for j in range(low, high):
        if nums[j] > pivot:
            nums[i], nums[j] = nums[j], nums[i]
            i += 1
    nums[i], nums[high] = nums[high], nums[i]
    return i
```

Introduction - Motivation

- Most current benchmarks for code only focus on the functional correctness of code generated by LLMs or their ability to understand text and code.



Benchmarks

These leaderboards are used to track progress in Code Generation

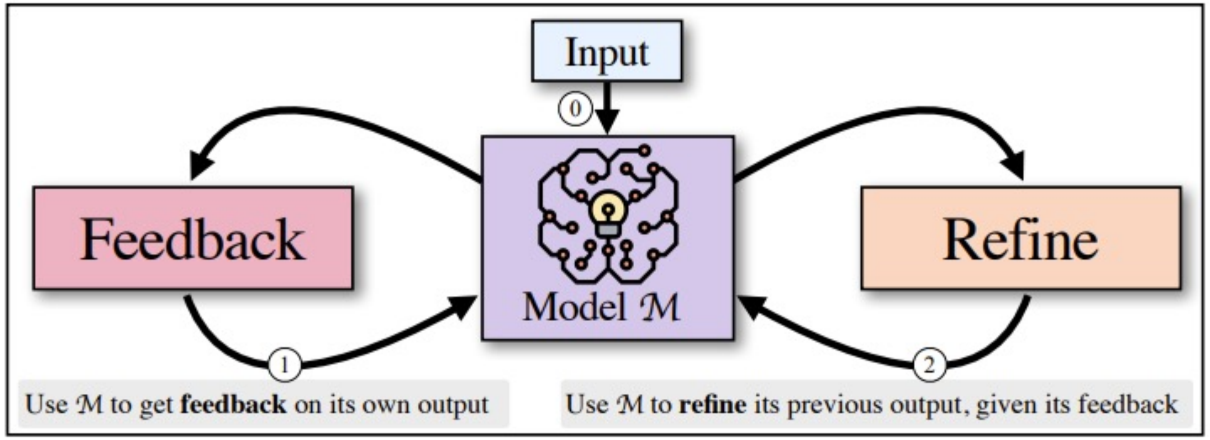
Trend	Dataset	Best Model
	HumanEval	Language Agent Tree Search (GPT-4)
	CoNaLa	PanGu-Coder-FT-I
	WikiSQL	NL2SQL-RULE
	APPS	CodeRL+CodeT5
	Django	MarianCG
	CoNaLa-Ext	BART W/ Mined
	MBPP	Language Agent Tree Search (GPT-3.5)
	Shellcode_IA32	CodeBERT

Introduction - TimeEval

- Dataset
  - Problem set of size 110
  - Canonical solution for each problem
  - Test cases for each question
- A framework for automated measurement of code efficiency

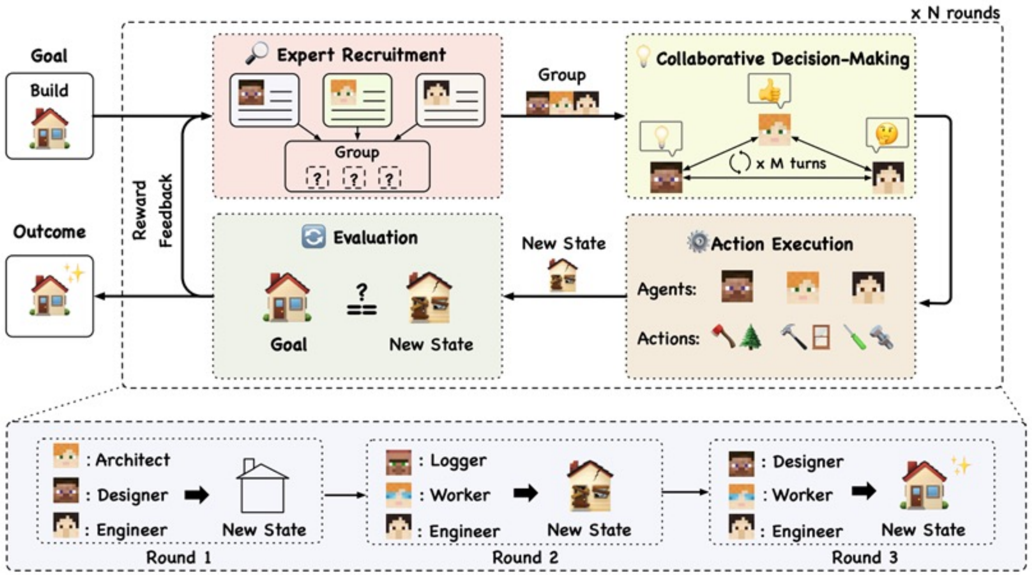
Background

- Self-refinement:
  a framework aiming to imitate the process of human thinking

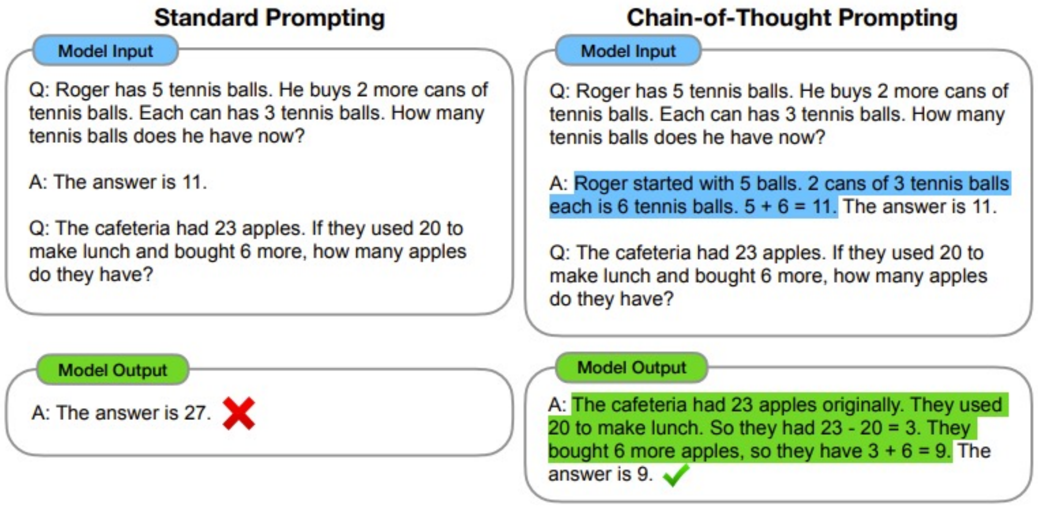


Background

- LLM-based Multi-Agents Collaboration



- In-context Learning
  - Zero-shot
  - One-shot / Few-shot
  - Chain of Thought



Dataset Processing & Benchmark Creation

Dataset Processing

APPS dataset

- 10,000 coding problem in total.
- The average number of test cases for each problem is 21.2.
- 232,444 ground truth solutions written by human.

Dataset Processing & Benchmark Creation

Dataset Processing

APPS dataset → TimeEval dataset

- **110 High quality questions**: The problem set comprises 110 questions designed to assess the efficiency of generated code.
- **Canonical solution for each problem**: We provide an optimal solution for each problem.
- **Test cases for each question**: We prepared ten test cases containing both small and large sizes for each question.

Dataset Processing & Benchmark Creation

Dataset Processing

Dataset File Structure

```
● (base) canranliu@Canrans-MacBook-Pro data % tree 0032
0032
├── canonical_solution.py
├── input_output.json
├── metadata.json
└── question.txt

1 directory, 4 files
```

Dataset Processing & Benchmark Creation

Dataset Processing

Question 0032 in our dataset

data > 0032 > question.txt

You, 1 second ago | 1 author (You)

```
We consider a positive integer perfect, if and only if the sum of its digits is exactly $10$. Given a positive integer $k$, your task is to find the $k$-th smallest perfect positive integer.


-----Input-----

A single line with a positive integer $k$ ($1 \leq k \leq 10\,000$).


-----Output-----

A single number, denoting the $k$-th smallest perfect integer.


-----Examples-----
Input
1

Output
19

Input
2

Output
28



-----Note-----

The first perfect integer is $19$ and the second one is $28$.

```

question.txt

Dataset Processing & Benchmark Creation

Dataset Processing

Question 0032 in our dataset

data > 0032 > canonical_solution.py > ...

```python
import sys

#f = open('input', 'r')
f = sys.stdin

d = [[0] * 11 for _ in range(11)]

d[0][0] = 1
for i in range(10):
    for j in range(11):
        for k in range(10):
            if j+k <= 10:
                d[i+1][j+k] += d[i][j]

target = int(f.readline())
tt=target
target -= 1
val = 10
ans = ''
for i in range(10):
    ii=9-i
    for j in range(val+1):
        if j==10:
            continue
        jj=val-j
        if d[ii][jj] <= target:
            target -= d[ii][jj]
        else:
            val = jj
            ans += str(j)
            break
print(int(ans))
```

canonical_solution.py

Dataset Processing & Benchmark Creation

Dataset Processing

Question 0032 in our dataset

```
data > 0032 > {} input_output.json > ...
{
    "inputs":  ["1\n",        "2\n",        "13\n",      "101\n",       "1023\n",
                "9999\n",     "10000\n",    "2333\n",    "9139\n",      "9859\n"],

    "outputs": ["19\n",       "28\n",       "136\n",     "1432\n",      "100270\n",
                "10800010\n", "10800100\n", "310060\n",  "10134010\n",  "10422001\n"]
}
```

input_output.json

```
data > 0032 > {} metadata.json > ...
{
    "difficulty": "interview",
    "url": "https://codeforces.com/problemset/problem/919/B",
    "time complexity" : "O(1)"
}
```

metadata.json

Dataset Processing & Benchmark Creation

Benchmark Creation

Execute code

0002_result.txt

test_result > 0002_result.txt

```
canonical_solution.py:
    Results: ['True', 'True', 'True', 'True', 'True', 'True', 'True', 'True', 'True', 'True']
    Outputs: ['2\n', '9\n', '0\n', '31\n', '318140\n', '0\n', '2044\n', '296190217\n', '235\n', '199999823\n']
    Passed tests: 10
    Wrong answers: 0
    Time limit exceeded: 0
    Execution times: ['0.014', '0.013', '0.013', '0.013', '0.013', '0.013', '0.013', '0.013', '0.013', '0.013']
    Total time: 0.13 seconds

gen_solution.py:
    Results: ['True', 'True', 'True', 'True', 'True', 'True', 'True', 'Timeout', 'True', 'Timeout']
    Outputs: ['2\n', '9\n', '0\n', '31\n', '318140\n', '0\n', '2044\n', 'Timeout', '235\n', 'Timeout']
    Passed tests: 8
    Wrong answers: 0
    Time limit exceeded: 2
    Execution times: ['0.014', '0.013', '0.013', '0.014', '0.280', '0.013', '0.014', 'Timeout', '0.015', 'Timeout']
    Total time: 10.38 seconds
```

Benchmark Creation

Metrics

- **Pass rate**: Percentage of test cases that passed the test out of all test cases.
- **Fail rate**: Percentage of test cases that failed the test out of all test cases.
- **Timeout rate**: Percentage of timeout test cases out of all test cases.

Dataset Processing & Benchmark Creation

Benchmark Creation

- **Percent Optimized**: %Opt

Metrics

- **Percent Optimized** %Opt: Proportion of programs where the execution time of the generated code is close enough to the execution time of the optimal solution in the test set (Canonical solution). That is, the code execution time that satisfies the equation,

$$\frac{t_{\text{gen}} - t_{\text{opt}}}{t_{\text{opt}}} < \theta$$

where t_{gen} represents the execution time of generated code, t_{opt} represents the execution time of optimal code and θ represents the threshold. The execution time is defined as close enough when the LHS is less than the threshold.

Dataset Processing & Benchmark Creation

Benchmark Creation

- **Speedup**: %Sp

Metrics

Speedup %Sp: The ratio of the execution time of the optimal solution to the execution time of the generated program. This metric accurately describes how close in time the generated program is to the optimal solution.

$$SPEEDUP = \frac{t_{\text{opt}}}{t_{\text{gen}}}$$

Dataset Processing & Benchmark Creation

Benchmark Creation

Baseline

	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
gpt-3.5-turbo	68.5	1.6	29.8	0.0	8.3

Research Questions

- RQ1: Does self-refinement improve the efficiency of generated code?
- RQ2: How to enhance the refinement result when using the self-refinement technique.
- RQ3:Does the Multi-agent collaboration technique improve the efficiency of generated code?
- RQ4: How different assignments of roles to agents and different collaborative structures will affect results.
- RQ5: The impact of in-context learning on the efficiency of generated code.
- RQ6: The effect of other parameters or LLM types on the efficiency of generated code.

Experiment: Self-Refinement

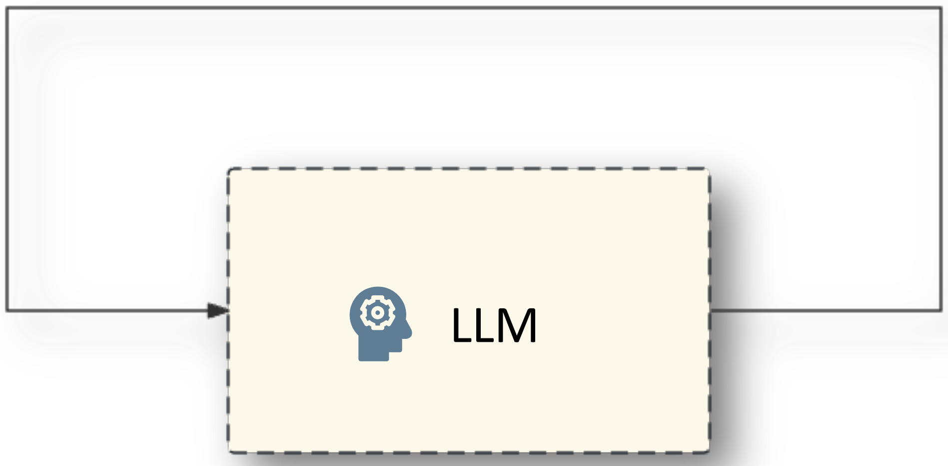
RQ1: Does self-refinement improve the efficiency of generated code?

When sample size is the same

Self-Refine

VS

Best of K

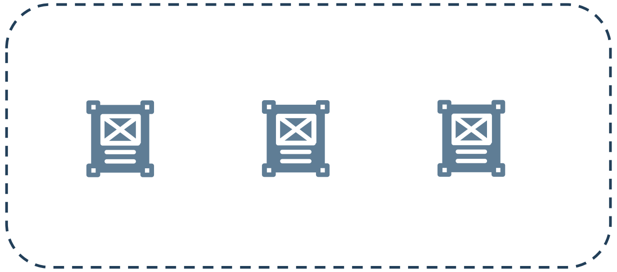


Original code

1st round code

2nd round code

VS



Best of them

Experiment: Self-Refinement

RQ1: Does self-refinement improve the efficiency of generated code?

Self-Refine

VS

Best of K

Round	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
0	68.5	1.6	29.8	0.0	8.3
1	61.3	17.0	21.6	20.0	32.5
2	67.8	5.2	27.0	4.5	12.9
3	66.6	6.7	26.6	3.6	12.6

K	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
1	68.5	1.6	29.8	0.0	8.3
2	67.8	3.4	28.8	0.9	9.7
3	67.7	3.3	29.0	1.8	10.0
4	66.9	4.3	28.7	3.6	14.3

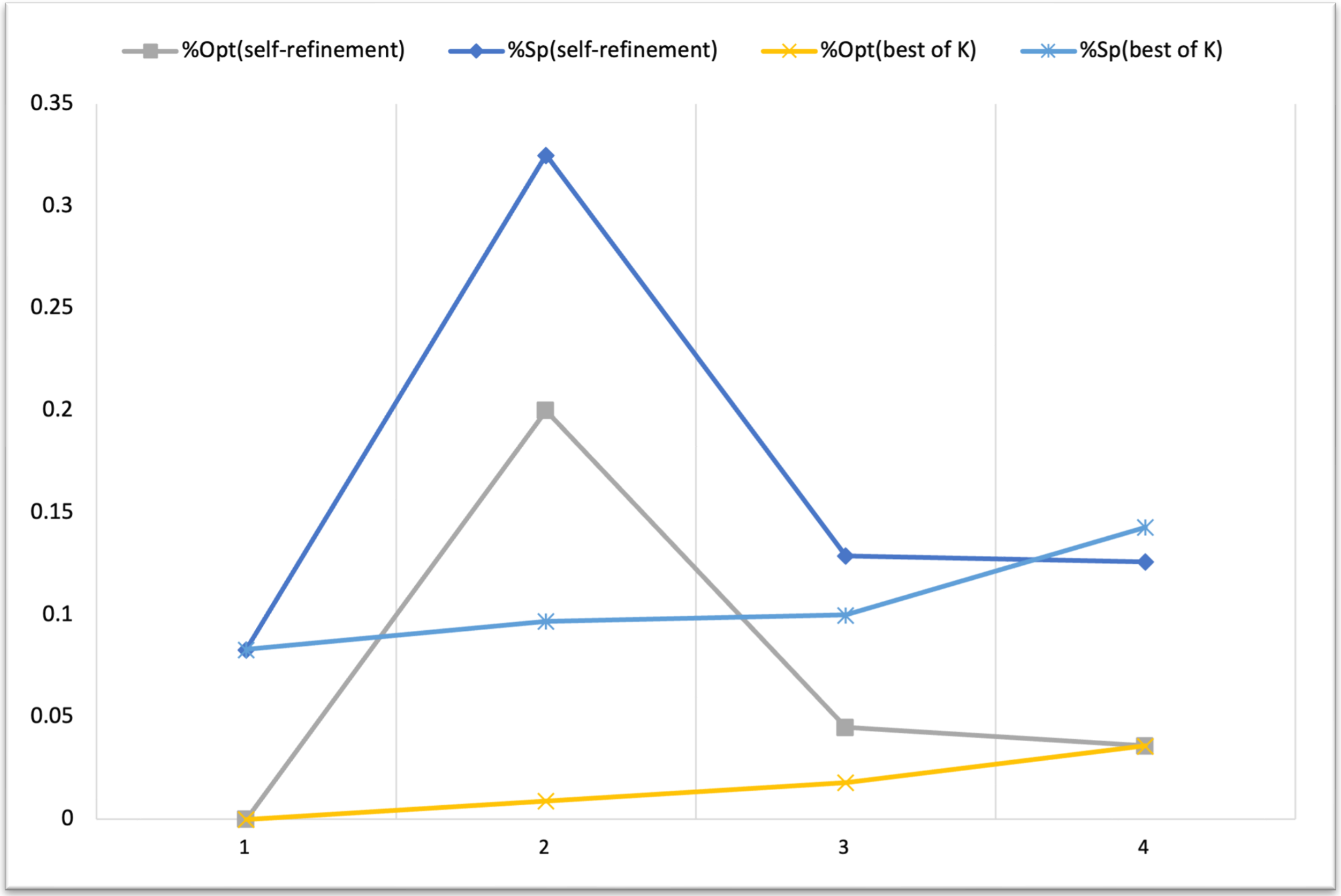
Experiment: Self-Refinement

RQ1: Does self-refinement improve the efficiency of generated code?

Self-Refine

VS

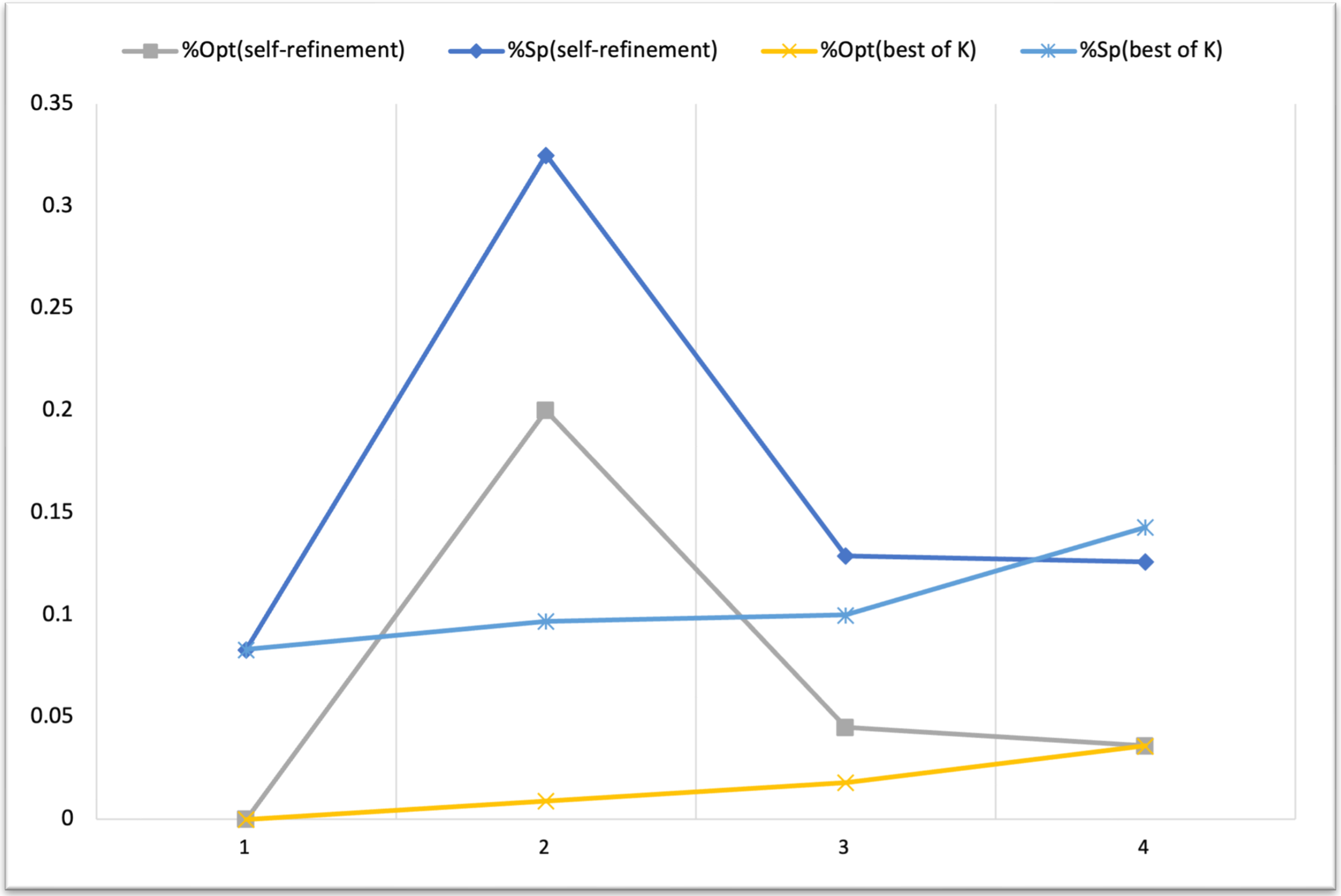
Best of K



Experiment: Self-Refinement

RQ2: How to enhance the refinement result when using the self-refinement technique.

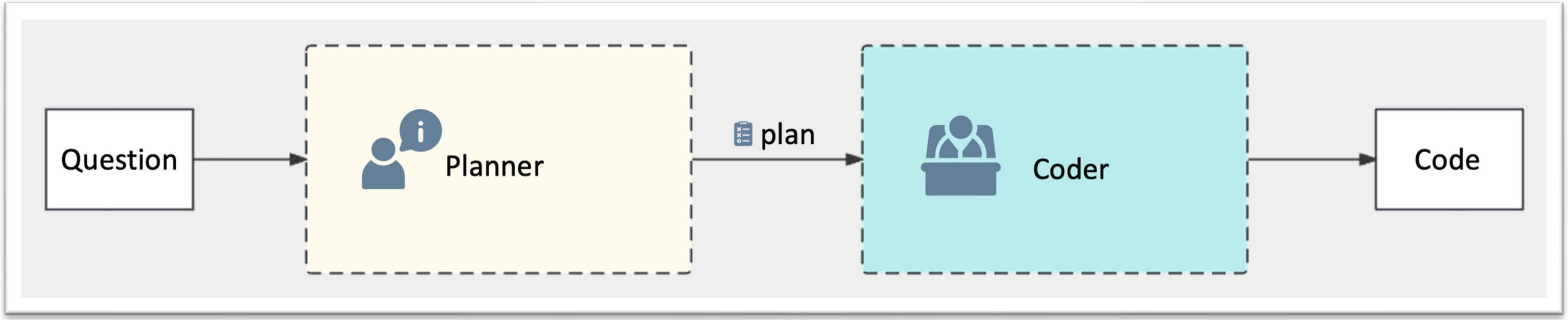
Number of Round



Experiment: Multi-Agent Collaboration

RQ3: Does the Multi-agent collaboration technique improve the efficiency of generated code?

Planner + Coder



Experiment: Multi-Agent Collaboration

RQ3: Does the Multi-agent collaboration technique improve the efficiency of generated code?

Plan

```
To solve this problem, we can use a brute force approach to check all possible combinations of side lengths a, b, and c. However, this approach would have a time complexity of O(n^3), which would be too slow for large values of n.

Instead, we can optimize the solution by using a more efficient approach. We can iterate over the values of a and b, and for each combination, calculate the value of c using the Pythagorean equation. If the calculated value of c is less than or equal to n, we can increment the count of valid triangles.

Here is a plan to solve the problem with low time complexity:

1. Read the input value of n.
2. Initialize a variable count to 0 to keep track of the number of valid triangles.
3. Iterate over the values of a from 1 to n.
   - For each value of a, iterate over the values of b from a to n.
     - For each combination of a and b, calculate the value of c using the Pythagorean equation: c = sqrt(a^2 + b^2).
     - If the calculated value of c is less than or equal to n, increment the count of valid triangles.
4. Print the value of count as the output.

By following this plan, we can solve the problem with a time complexity of O(n^2), which is much more efficient than the brute force approach.
```

Experiment: Multi-Agent Collaboration

RQ3: Does the Multi-agent collaboration technique improve the efficiency of generated code?

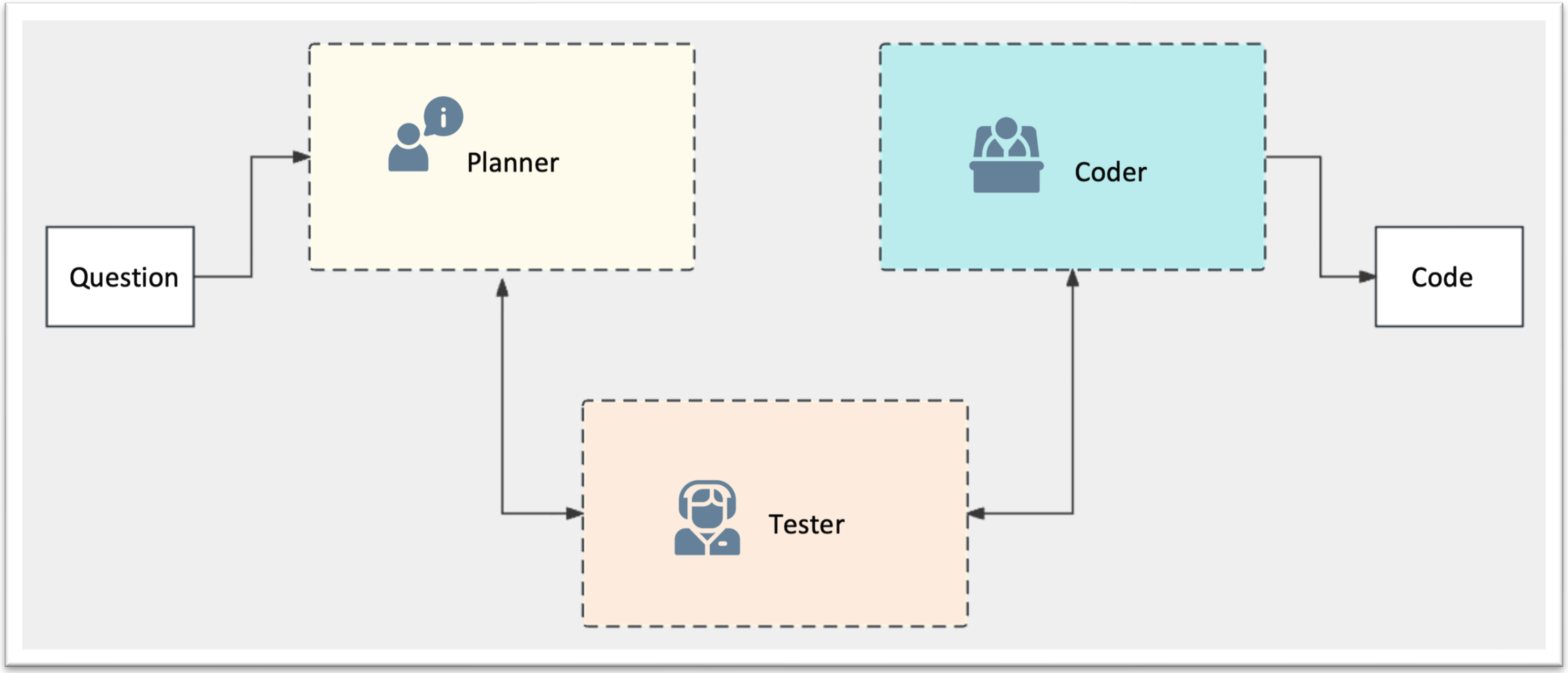
Planner + Coder

Agents	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
Coder	68.5	1.6	29.8	0.0	8.3
Planner + Coder	37.1	44.5	18.2	29.0	46.1

Experiment: Multi-Agent Collaboration

RQ4: How different assignments of roles to agents and different collaborative structures will affect results.

Planner + Coder +Tester




```
while(!Tester.isCorrect(plan)):
    issue = Tester.generate(plan)
    Tester.send(issue, Planner)
    plan = Planner.generate(issue)
    Planner.send(plan, Tester)
```

```
while(!Tester.isCorrect(code)):
    issue = Tester.generate(code)
    Tester.send(issue, Coder)
    code = Coder.generate(issue)
    Coder.send(code, Tester)
```

Experiment: Multi-Agent Collaboration

RQ4: How different assignments of roles to agents and different collaborative structures will affect results.

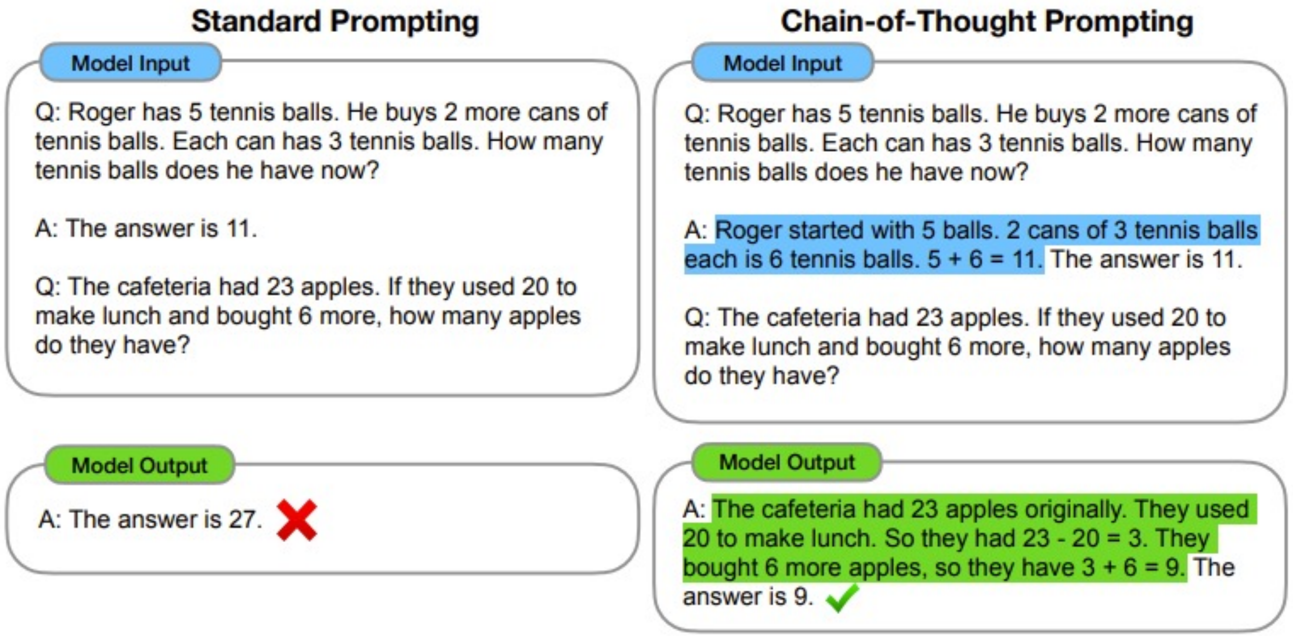
Planner + Coder +Tester

Agents	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
Coder	68.5	1.6	29.8	0.0	8.3
Planner + Coder	37.1	44.5	18.2	29.0	46.1
Planner + Coder +Tester	55.6	26.6	17.7	29.1	46.9

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

In context Learning
- Zero-shot
- One-shot / Few-shot
- Chain of Thought



Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Zero-shot

Prompt format:

Problem: problem from our dataset

```python
def get_messages(problem):
    messages = []
    messages.append(
        {"role": "system", "content": "In the following question you only need to generate the code itself, " +
         "and directly generate a python code that can be run directly. Don't add any text descriptions!"}
    )
    messages.append(
        {"role": "user", "content": problem}
    )

    return messages
```

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

One-shot

Prompt format:

Problem: problem of the example;
Solution with poor time complexity;
Solution with good time complexity;
} example

Problem: problem from our dataset:
Please provide a solution with good time complexity.

```python
def get_messages(problem):
    messages = []
    messages.append(
        {"role": "system", "content":
         "Problem: You are given a 0-indexed array of integers nums of length n.
         + "Solution with poor time complexity:\ndef jump(self, nums: List[int])
         + "Solution with good time complexity:\ndef jump(self, nums: List[int])
    )
    messages.append(
        {"role": "user", "content": f"Problem: {problem}
         Please provide a solution with good time complexity.
         Please directly provide a python code that can be run directly.
         Don't add any text descriptions!"}
    )

    return messages
```

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Zero-shot vs One-shot

Experiment	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
Zero-shot	67.0	1.6	31.4	1.8	11.9
One-shot	56.6	23.4	20.0	22.7	37.2

- %Opt and %Sp metrics improved
  --> the result of one-shot is closer to the optimal solution than the zero-shot
- The accuracy decreased
  --> one-shot led the model to focus more on the time complexity of the code

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Self-refinement
+
One-shot

Prompt format:

Problem: problem of the example;
Original solution;
Solution with improved time complexity;

example

Problem: problem from our dataset;
Original solution: baseline solution;
Please provide a solution with improved time complexity.

```
def get_messages(prompt, original_solution):
    messages = []
    messages.append(
        {"role": "system", "content": "Problem: You are given a 0-indexed array of integers nums of length n.
         + "Original solution:def jump(self, nums: List[int]) -> int:\n    size = len(nums)\n    dp = [float(
         +  "Solution with improved time complexity:\ndef jump(self, nums: List[int]) -> int:\n    end, max_p
    )
    messages.append(
        {"role": "user", "content": f"Problem: {prompt}
         Original solution: {original_solution}
         Please provide a solution with improved time complexity.
         Please directly provide a python code that can be run directly. Don't add any text descriptions!"}
    )
```

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Self-refinement
+
One-shot

Experiment	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
Zero-shot	67.0	1.6	31.4	1.8	11.9
One-shot	56.6	23.4	20.0	22.7	37.2
Self-refinement + One-shot	**58.9**	**22**	**19.1**	**25.5**	**35.4**

- This experiment did not show a significant improvement over simple self-refinement or one-shot learning alone
- The accuracy is the highest among three experiments, but both %Opt and %Sp metrics were at intermediate values

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Self-refinement
+
One-shot
+
CoT

Thought process:

- What is the time complexity of the original solution?
- Is there a better algorithm in terms of time complexity?
- What is the time complexity of this algorithm?
- How to implement this algorithm?

```
def get_messages(prompt, original_solution):
    messages = []
    messages.append(
        {"role": "system", "content": "Problem: You are given a 0-indexed array of integers nums 
         + "Original solution:def jump(self, nums: List[int]) -> int:\n    size = len(nums)\n    
         + "How to improve the time complexity?\n"
         + "Answer:The time complexity of the original solution is O(n^2) due to the nested loops
         + "So the solution with improved time complexity is:\ndef jump(self, nums: List[int]) ->
    )
    messages.append(
        {"role": "user", "content": f"Problem: {prompt} \n
         Original solution: {original_solution} \n
         How to improve the time complexity?
         Answer:"}
    )
```

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Self-refinement
+
One-shot
+
CoT

Experiment	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
Zero-shot	67.0	1.6	31.4	1.8	11.9
One-shot	56.6	23.4	20.0	22.7	37.2
Self-refinement + One-shot	58.9	22	19.1	25.5	35.4
Self-refinement + One-shot +CoT	**35.8**	**55.4**	**8.8**	**60.0**	**84.8**

- Significant improvements in both %Opt and %Sp metrics
- But also significant decrease in accuracy

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Self-refinement

+

One-shot

+

CoT

+

Test cases

```
def get_messages(prompt, original_solution, input_output):
    messages = []
    messages.append(
        {"role": "system", "content": "Problem: You are given a 0-indexed array of integers nums of lengt
         + "Original solution:def jump(self, nums: List[int]) -> int:\n    size = len(nums)\n    dp = [fl
         + "How to improve the time complexity?\nAnswer:The time complexity of the original solution is O
         + "So the solution with improved time complexity is:\ndef jump(self, nums: List[int]) -> int:\n
    )
    messages.append(
        {"role": "user", "content": f"Problem: {prompt} \n Original solution: {original_solution} \n
         How to improve the time complexity? Please also pay attension to the accuracy according to
         these inputs and correct outputs of test cases and outputs of the original code: {input_output}
         Answer:(Please mark the start and and of python script)"}
    )
```

Experiment: In-Context Learning

RQ5: The Impact of In-Context Learning on the Efficiency of Generated Code.

Self-refinement + One-shot
+
CoT + Test cases

Experiment	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
Zero-shot	67.0	1.6	31.4	1.8	11.9
One-shot	56.6	23.4	20.0	22.7	37.2
Self-refinement + One-shot	58.9	22	19.1	25.5	35.4
Self-refinement + One-shot +CoT	35.8	55.4	8.8	60.0	84.8
Self-refinement + One-shot +CoT + Test cases	**40.8**	**49.9**	**9.3**	**53.6**	**72.5**

- There is indeed an improvement in accuracy, although it is still slightly lower compared to previous experiments
- Continue to explore how to further enhance accuracy in the future

Experiment: Temperature

RQ6: The effect of other parameters or LLM types on the efficiency of generated code.

Temperature	Temperature is a parameter provided by OpenAI for user adjustment. The choice of sampling temperature ranges from 0 to 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic.[1]

[1]https://platform.openai.com/docs/guides/code

Experiment: Temperature

RQ6: The effect of other parameters or LLM types on the efficiency of generated code.

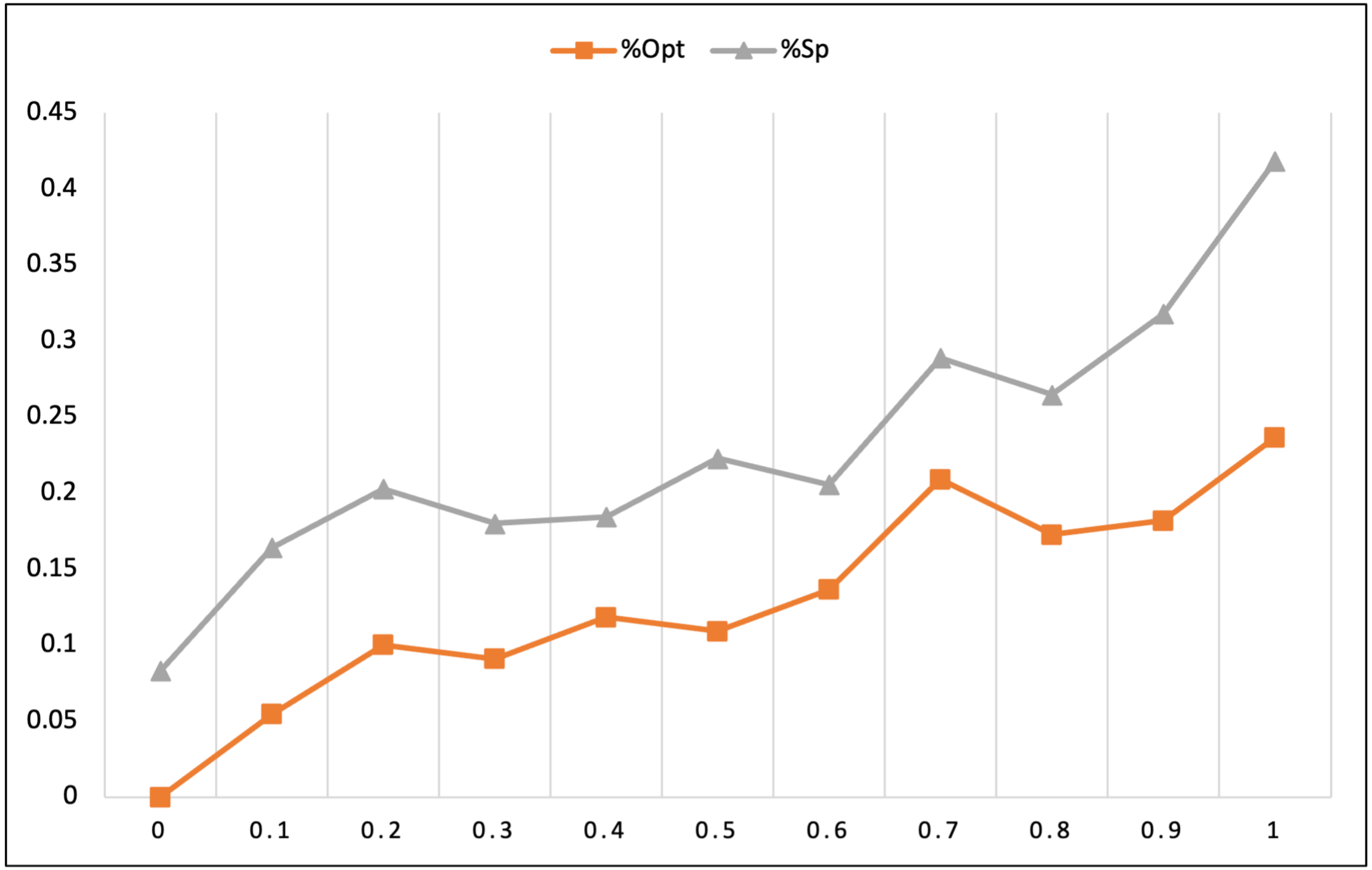
Temperature in the range of [0,1]

Temperature	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
0.0	68.5	1.6	29.8	0.0	8.3
0.1	67.6	5.3	27.1	5.4	16.3
0.2	65.3	10.2	24.5	10.0	20.3
0.3	65.5	9.4	25.2	9.1	18.0
0.4	60.3	13.3	26.5	11.8	18.4
0.5	59.5	14.5	26.0	10.9	22.3
0.6	58.7	15.9	25.4	13.6	20.5
0.7	54.5	20.8	24.6	20.9	28.9
0.8	55.7	18.4	25.9	17.3	26.5
0.9	53.1	25.3	21.6	18.2	31.8
1.0	46.9	31.4	21.7	23.6	41.8

Experiment: Temperature

RQ6: The effect of other parameters or LLM types on the efficiency of generated code.

Temperature in the range of [0,1]



Experiment: Comparison of Different LLMs

RQ6: The effect of other parameters or LLM types on the efficiency of generated code.

gpt-3.5-turbo

vs

gpt-4

Model	Pass Rate	Wrong Rate	Timeout Rate	%Opt	%Sp
gpt-3.5-turbo	68.5	1.6	29.8	0.0	8.3
gpt-4	61.3	17.0	21.6	20.0	32.5

Summary of Our Contributions

- Proposed timeEval benchmark.

- On our benchmark, we did empirical studies of the existing models or frameworks to test the efficiency of generated code.

- Proposed several frameworks to improve the efficiency of generated code.

Future Works

- Continue to measure the different models as well as the framework on our benchmark.

- Try to create a more efficient framework.

- Begin an exploration of the space complexity of the generated code.

Q&A Session

Thank you