POSTECH

CODEI/O: Condensing Reasoning Patterns via Code Input-Output Prediction

Kang Minseo, Choi Won Joon, Baek Seong-Eun

Introduction

Reasoning

Role of reasoning

- A fundamental aspect of human cognition and problem-solving
- Enables **fast adaptation and transfer** to new tasks
- Cornerstone of advanced LLMs
- A critical step toward Artificial General Intelligence (AGI)

Current Challenge

- Math problem solving and code generation benefit from **abundant structured data**
- But:
  - Logical deduction
  - Scientific inference
  - Symbolic reasoning

  suffer from **sparse, fragmented supervision signals**

✓ It becomes crucial to identify training data that is rich in **diverse reasoning patterns** while also being **scalable** to obtain.

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Why Existing Methods Fall Short

Conventional continual pre-training

- Relevant reasoning signals are often implicit
- Intertwined with noisy information

✓ Suboptimal

Text-to-code generation

- It requires the generation of code specific syntax
- This limitation makes it difficult to generalize to non-code tasks

✓ Also faces challenges

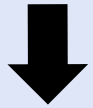
Proposed method

- Transforming raw code files into executable functions and framing a straightforward task
- Given a function and a textual query, the model predicts either the output from a given input or the input from a given output

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Proposed method

Function + Query



Input → Predict Output (CoT)

OR

Output → Predict Input (CoT)

- The prediction is expressed as a Chain-of-Thought (CoT) in natural language

- By collecting and transforming functions from diverse sources, the resulting data captures a variety of foundational reasoning skills
  - logic flow orchestration
  - state-space exploration
  - recursive decomposition
  - decision-making

- Helps internalize core reasoning skills more effectively

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Input/Output Prediction as a Pretraining Stage

Our code input/output prediction learning

- Positioned before general instruction tuning
- Intermediate step to enhance the reasoning abilities of the base model

Prompt Format

- Each prompt includes:
  - -A function
  - -A textual query
  - -A given input or output

Data Collection Process

- Responses generated by prompting DeepSeek-V2.5
- Highly scalable:
  - - Sample hundreds of inputs per function
  - - Execute each funtion to collect ground-truth outputs

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

CODEI/O & CODEI/O++

CODEI/O Data

- Collected **450K+ functions** from multiple sources
- For each function, **multiple input-output pairs are generated** by executing the code
- **CoTs** (Chain-of-Thoughts) are synthesized for these I/O pairs

✓ Result: 3.5 million training samples → **CODEI/O data**

Refinement Process: CODEI/O++

- Leverages the verifiable nature of code: All predictions are verified by executing the code
- **Incorrect predictions are revised** by DeepSeek-V2.5 again (multi-turn revision)

✓ Resulting in: **CODEI/O++**, which further improves model performance

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Consistent & Generalized Reasoning

Validation

- Validated on 4 base models (sizes: 7B to 30B)
- Assessments across 14 different benchmarks, including:
  - Code-related, logic, symbolic, mathematical & numerical, scientific, commonsense

Compared to Strong Data Baselines

- OpenMathInstruct2, OpenCoderSFT-Stage1, WebInstruct, High-quality raw code

✓ **Higher average scores** across all base models

✓ **More balanced performance** across most benchmarks:

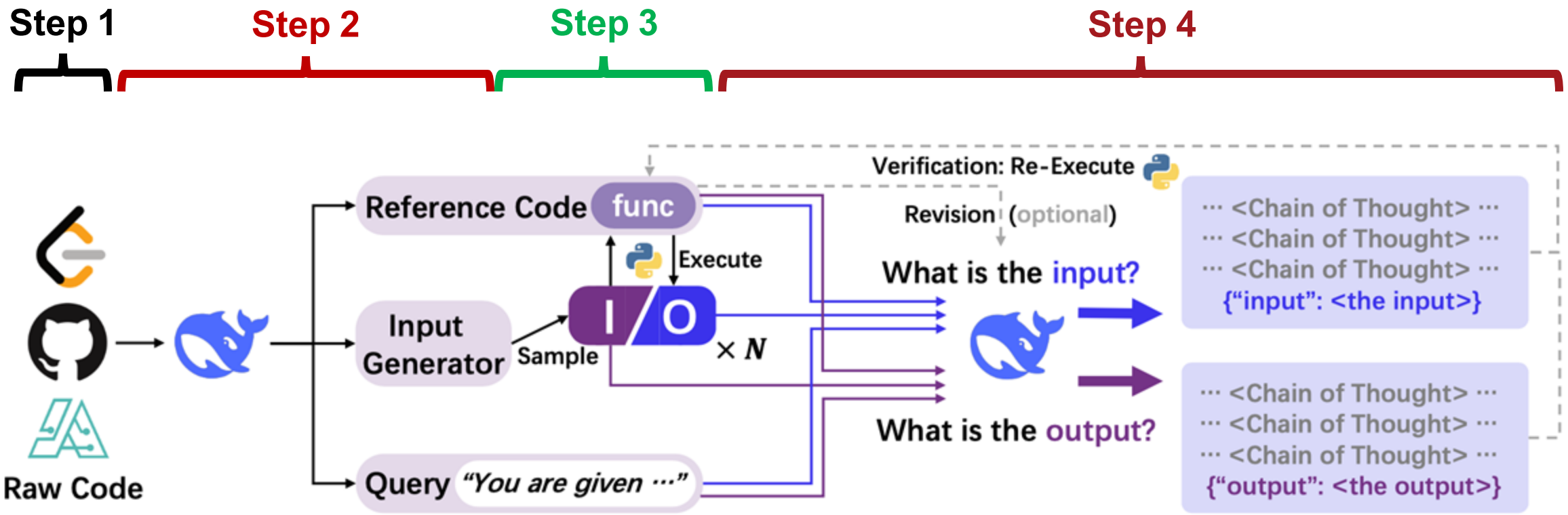
- CODEI/O offers consistent improvements across nearly all benchmarks.

✓ **“ Balanced and generalizable reasoning abilities ”**

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Code I/O

CODE I/O



Overview of dataset construction

CODE I/O : Dataset Construction

Step 1: Collecting raw code files

To obtain diverse reasoning patterns, codes should be obtained from various sources.
- Raw code sources to construct CODE I/O are as below:
 1) **CodeMix**: In-house raw python code for pre-training the LLM (DeepSeek).
 2) **PyEdu-R**: Python codes for complex reasoning tasks on STEM or logic puzzles.
 3) Other well-known Github public repositories/online platforms.

In total, approximately 810.5K raw codes were obtained.

CODE I/O : Dataset Construction

Step 2: Transforming to a unified format

The collected raw codes often lack structure, contain irrelevant parts.
➔ An LLM(DeepSeek-V2.5) was utilized for the pre-processing.

The formatted data have 4 parts.

1) Cleaned reference code (with main entrypoint function)

- Non-essential code parts are filtered (e.g. print, plot)
- Main entrypoint function: overall logic of the raw code

2) Natural language query

- Problem statement for the main entrypoint function.

3) Input/output format description

4) Input random generator

```
#get the vertical acceleration data
acceleration [. . . . . ]
# pass acceleration data to low pass filter
time[. . . . . ]

#code to find speed at each point
initial_speed = current_speed
delta_t = t_current  - t_prev
curr_acc = acc[i]
current_speed = initial_speed + (curr_acc) *
     delta_t

#code to find dispacement

initial_dis = current_disp
delta_t = t_current  - t_prev
curr_speed = ("call above algorithm")
current_disp = initial_dis + (initial_speed +
     current_speed)/2 * delta_t ;
```

Raw code example

Cleaned Reference Code (with Main Entrypoint Function)

```
# import necessary packages
import numpy as np

# main function
def main_solution(acceleration, time, initial_speed
    , initial_displacement):
    # Convert inputs to numpy arrays if they are
        not already
    acceleration = np.array(acceleration)
    time = np.array(time)

    # Initialize variables
    current_speed = initial_speed
    current_disp = initial_displacement

    # Calculate speed and displacement
    speeds = []
    displacements = []

    for i in range(1, len(time)):
        delta_t = time[i] - time[i-1]
        curr_acc = acceleration[i]
        current_speed = current_speed + curr_acc *
            delta_t
        speeds.append(current_speed)

        current_disp = current_disp + (
            initial_speed + current_speed) / 2 *
            delta_t
        displacements.append(current_disp)

        initial_speed = current_speed

    # Convert outputs to JSON serializable format
    speeds = [float(speed) for speed in speeds]
    displacements = [float(disp) for disp in
        displacements]

    return {"speeds": speeds, "displacements":
        displacements}
```

Query

Given a set of vertical acceleration data and corresponding time points, how can we determine the speed and displacement of a vehicle at each time point, starting from an initial speed and displacement?

Input/Output Description

```
Input:
    acceleration (list of float): List of vertical
        acceleration values at each time point.
    time (list of float): List of time points
        corresponding to the acceleration values.
    initial_speed (float): Initial speed at the
        first time point.
    initial_displacement (float): Initial
        displacement at the first time point.

Output:
    return (dict): A dictionary with two keys:
    - speeds (list of float): List of calculated
        speeds at each time point.
    - displacements (list of float): List of
        calculated displacements at each time
        point.
```

Input Generator

```python
def input_generator():
    # Generate random acceleration data
    acceleration = [np.random.uniform(-10, 10) for
        _ in range(10)]

    # Generate corresponding time data
    time = [0.1 * i for i in range(10)]

    # Generate initial speed and displacement
    initial_speed = np.random.uniform(0, 10)
    initial_displacement = np.random.uniform(0, 10)

    return {
        "acceleration": acceleration,
        "time": time,
        "initial_speed": initial_speed,
        "initial_displacement":
            initial_displacement
    }
```

CODE I/O : Dataset Construction

Step 3: Input/output pair collection

- Using the cleaned reference code & the input random generator, obtain Input/output value pairs for each code.

- While data collection, the cases below are excluded:
 1) The reference code has randomness (Input/output pairs should be deterministic)
 2) Too much runtime is required
 3) Input/output structure is too complicated
 4) The reference code fails to run

After filtering out, 3.5M instances were obtained from 454.9K raw code files. (50% for input prediction, 50% for output prediction)

CODE I/O : Dataset Construction

Step 4: Building samples for input-output prediction

- Using the collected instances, generate labels for the dataset.
- The label should contain two components:
  1) Reasoning flow in natural language (CoT).
  2) Predicted input(output) value from the given output(input) value.

- Obtaining labels with fixed templates would be reliable. However, this approach has critical limitations.
  1) Cannot build deterministic reverse functions for all codes.
  2) The label style sticks, which would harm the generalization ability.

➔ Due to the limitations, labels are fully generated by the LLM (DeepSeek-V2.5).

Query

You are given an amount of money `amt` and a list of coin denominations `coins`. Your task is to determine the minimum number of coins needed to make up the amount `amt` using the given denominations. If it is not possible to make the amount with the given coins, the function should return `inf` (infinity).

Reference Code

```
def change_ref(amt, coins):
    if amt <= 0: return 0
    if amt != 0 and not coins: return float("inf")
    elif coins[0] > amt: return change_ref(amt,
coins[1:])
    else:
        use_it = 1 + change_ref(amt - coins[0],
coins)
        lose_it = change_ref(amt, coins[1:])
        return min(use_it, lose_it)
```

output prediction label

Given input = {"amt": 25, "coins": [1,4,7]}, predict output

1. Greedy Approach: We want to use as many of the largest denomination coins as possible to minimize the number of coins. However, the greedy approach doesn't always work for all coin systems, so we need to consider all possible combinations.

2. Possible Combinations:
   - Using the largest coin `7`:
     - We can use `7` coins up to `3` times (since `3 * 7 = 21`).
     - After using `3` coins of `7`, we have `25 - 21 = 4` left.
     - To make `4`, we can use `1` coin of `4`.
     - Total coins used: `3 (for 7) + 1 (for 4) = 4` coins.

   - Using fewer `7` coins:
     - If we use `2` coins of `7`, we have `25 - 14 = 11` left.
     - To make `11`, we can use `2` coins of `4` (totaling `8`) and `3` coins of `1`.
     - Total coins used: `2 (for 7) + 2 (for 4) + 3 (for 1) = 7` coins.

   - Using `1` coin of `7`:
     - We have `25 - 7 = 18` left.
     - To make `18`, we can use `4` coins of `4` (totaling `16`) and `2` coins of `1`.
     - Total coins used: `1 (for 7) + 4 (for 4) + 2 (for 1) = 7` coins.

   - Using no `7` coins:
     - We can use `6` coins of `4` (totaling `24`) and `1` coin of `1`.
     - Total coins used: `6 (for 4) + 1 (for 1) = 7` coins.

3. Optimal Solution: The combination that uses the fewest coins is the one where we use `3` coins of `7` and `1` coin of `4`, totaling `4` coins.
{"output": 4}

Input prediction label

Given output = 4, predict input

1. Objective: We need to find an input where the minimum number of coins required to make up the amount `amt` is `4`.
2. Coin Denominations: The list of coin denominations should be such that the combination of coins that sums up to `amt` requires exactly `4` coins.
3. Example: Suppose we have coin denominations `[1, 2, 5]` and we want to make up the amount `8`. The optimal way to make up `8` using these coins is:
   - 1 coin of `5`
   - 1 coin of `2`
   - 1 coin of `1`
   - This sums up to `8` using `3` coins, which is less than `4`. So, this is not the correct input.
4. Another Example: Let's consider coin denominations `[1, 3, 4]` and the amount `6`. The optimal way to make up `6` using these coins is:
   - 1 coin of `4`
   - 2 coins of `1`
   - This sums up to `6` using `3` coins, which is still less than `4`.
5. Correct Example: Now, consider coin denominations `[1, 3, 4]` and the amount `8`. The optimal way to make up `8` using these coins is:
   - 2 coins of `4`
   - This sums up to `8` using `2` coins, which is less than `4`.
6. Final Example: Let's try coin denominations `[1, 2, 5]` and the amount `13`. The optimal way to make up `13` using these coins is:
   - 2 coins of `5`
   - 1 coin of `2`
   - 1 coin of `1`
   - This sums up to `13` using `4` coins, which matches our desired output.

{"input": {"amt": 13, "coins": [1, 2, 5]}}

CODE I/O : Dataset Construction

Step 4: Building samples for input-output prediction

- To enhance the dataset quality, multi-turn approach was also applied.
- When the LLM predicts a wrong input/output value at 1st turn, human feedback is added to the prompt at the 2nd turn.

- The human feedback contains:
 1) Whether the predicted answer is correct
 2) Corresponding output of the wrong prediction in the 1st turn (only if the input prediction was wrong)
- The human feedback does not contain the ground truth value.

[Mismatch] Your input is not feasible! Given the output 4, your predicted input is {"target": 10, "numbers": [1, 2, 3, 4, 5]}, which actually gets a wrong output as 3

Human feedback example

CODE I/O++

Step 4: Building samples for input-output prediction

- Utilizing the LLM, two types of datasets were generated.

#1: CODE I/O [Single-turn dataset]

Prompt contains:
1) Cleaned reference code
2) A query
3) An input & output structure description
4) An input(or output) value

Label contains:
1) Turn 1 response from DeepSeek-V2.5

#2: CODE I/O++ [Multi-turn dataset]

Prompt contains:
Same as CODE I/O

Label contains:
1) Turn 1 response from DeepSeek-V2.5
2) Turn 1 feedback from human
3) Turn 2 response from DeepSeek-V2.5
4) Turn 2 feedback from human

Experiments

Experiment Setups

Base models: 4 types of LLMs are selected for experiments

Coder Models
- Qwen 2.5 7B Coder
- DeepSeek v2 Lite Coder

General-purpose Models
- LLaMA 3.1 8B
- Gemma 2 27B

Experiment Setups

Training procedure: two-step fine-tuning approach
Step 0) Prepare the baseline model
Step 1) Fine-tune by CODE I/O(++)
Step 2) Fine-tune by an extra instruction tuning dataset

*Instruction tuning dataset:
- Contains instruction/answer pairs from various domains & languages.
- Relatively smaller size than CODE I/O(++).
- Lets the model to acquire intent from the prompt.

Experiment Setups

To test whether CODE I/O can give general reasoning ability to LLMs, various evaluation benchmarks were used for the test:

BLUE : Math
GREEN: Code
RED: Logic reasoning
PURPLE: Deep domain knowledge

1) **DROP**: Numerical reasoning
2) **WinoGrande**: Commonsense reasoning
3) **GSM8K**: Basic arithmetic
4) **MATH**: Complex math reasoning
5) **MMLU-STEM**: Multiple-choice quizzes in STEM domain
6) **LeetCode-O**: Code output prediction
7) **BBH**: Challenging tasks for LLM models (logical deduction, boolean, …)
8) **GPQA**: Multiple-choice quizzes requiring deep domain knowledge (biology, chemistry, …)
9) **CruxEval**: Input/output prediction in python codes
10) **ZebraGrid**: Logic reasoning
11) **KorBench**: Logic reasoning
12) **LiveBench**: Logic reasoning & mathematics

Experiment Setups

To compare CODE I/O(++) to other works, four datasets were selected as comparison.

- **WebInstruct** (WI) (11.6M): Large instruction-tuning dataset mined from the Internet
- **OpenMathInstruct-2** (OMI2) (14.0M): Dataset focused on math-problem solving
- **OpenCoder-SFT-Stage-1** (OC-SFT-1) (4.2M): QA dataset generated from code data
- **Python-Edu** (PyEdu) (7.7M): Raw python code dataset

(For similar size as CODE I/O, 3.5M subsets of WI and OMI2 were used for experiments.)

Experiments - results

High perf. ⟵ ⟶ Low perf.

1st Stage Dataset	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
Qwen 2.5 Coder 7B																
2nd Stage Only		66.9	70.7	83.4	71.6	41.5	77.2	20.7	61.3	60.0	68.3	70.6	10.9	38.7	26.0	54.8
WI	3.5	66.3	73.5	87.0	71.4	39.1	77.5	18.3	59.1	61.6	68.6	68.7	10.2	42.5	26.0	55.0
WI (Full)	11.6	67.0	75.0	87.0	71.1	42.9	78.6	19.1	59.3	59.8	68.4	70.4	10.9	41.9	27.6	55.6
OMI2	3.5	67.6	74.3	84.1	72.3	36.2	77.4	20.9	60.4	61.5	68.8	69.3	10.1	42.7	27.2	55.2
OMI2 (Full)	14.0	66.9	74.0	88.5	73.2	40.9	77.8	19.9	59.5	62.4	68.3	71.3	11.2	41.2	28.4	56.0
OC-SFT-1	4.2	66.6	75.3	86.7	70.9	37.7	78.0	20.3	60.9	60.1	67.5	67.6	10.8	40.1	27.5	55.0
PyEdu	7.7	66.7	74.8	85.8	71.4	40.9	77.4	19.1	58.9	62.4	67.8	65.7	10.6	39.3	25.8	54.8
CodeI/O	3.5	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
CodeI/O++	3.5	66.9	79.1	85.7	72.1	40.6	77.9	24.2	62.5	67.9	71.0	74.2	10.7	45.7	29.1	57.7
LLaMA 3.1 8B																
2nd Stage Only		71.3	73.1	83.2	49.9	40.6	70.0	4.1	44.5	46.9	65.8	65.6	9.8	39.8	25.7	49.3
WI	3.5	72.1	76.3	82.8	52.8	42.9	69.6	4.1	44.0	44.8	64.5	67.8	10.0	42.7	23.1	49.8
OMI2	3.5	72.2	74.8	86.2	58.9	38.2	70.1	5.8	46.1	46.4	67.4	68.6	9.5	40.3	24.5	50.6
OC-SFT-1	4.2	71.0	71.9	81.8	51.1	38.2	68.4	5.7	43.5	44.9	65.6	67.6	10.5	42.0	24.7	49.1
PyEdu	7.7	70.6	69.6	83.2	49.8	42.4	69.1	5.2	43.1	44.5	64.0	65.6	10.2	42.6	25.7	49.0
CodeI/O	3.5	71.7	73.9	83.6	53.8	43.5	69.0	9.3	50.1	53.3	67.5	65.3	10.4	40.9	24.7	51.2
CodeI/O++	3.5	71.8	75.1	84.0	53.2	40.9	68.4	10.0	50.4	53.1	70.0	70.6	10.5	43.2	28.1	52.1

1st Stage Dataset	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
DeepSeek Coder v2 Lite 16B																
2nd Stage Only		68.4	73.4	82.5	60.0	38.6	68.5	14.8	53.0	54.9	61.1	69.2	6.7	44.7	26.6	51.6
WI	3.5	68.5	73.8	83.7	60.5	39.5	68.7	14.3	53.5	57.1	61.6	65.7	6.9	43.1	25.4	51.6
OMI2	3.5	67.6	74.1	84.7	64.7	38.4	70.1	14.4	53.8	55.8	63.6	66.4	6.4	42.0	24.7	51.9
OC-SFT-1	4.2	68.2	73.6	83.3	60.9	37.3	69.1	14.7	52.8	56.1	60.9	67.9	6.1	42.7	25.2	51.3
PyEdu	7.7	68.3	74.6	83.0	60.6	38.2	69.7	15.6	54.9	57.0	61.9	68.6	7.0	44.7	24.6	52.1
CodeI/O	3.5	68.4	74.6	83.6	60.9	38.6	70.3	18.7	58.4	62.8	63.1	70.8	7.8	46.0	26.1	53.6
CodeI/O++	3.5	69.0	73.5	82.8	60.9	38.8	70.0	20.3	59.5	61.0	64.2	69.4	6.7	46.3	26.9	53.5
Gemma 2 27B																
2nd Stage Only		72.4	80.1	90.1	66.3	44.4	82.8	19.1	62.5	66.9	77.1	80.4	13.5	47.8	30.0	59.5
WI	3.5	73.2	79.0	91.5	70.6	44.9	82.7	20.7	63.5	66.3	77.6	77.2	17.1	47.3	33.3	60.4
OMI2	3.5	73.1	79.3	90.8	67.1	44.0	83.4	19.2	61.4	66.0	77.1	80.5	13.9	49.7	40.7	60.4
OC-SFT-1	4.2	73.5	79.9	91.1	66.1	46.9	81.8	20.2	62.8	65.6	77.3	78.9	14.0	46.9	35.3	60.0
PyEdu	7.7	73.7	79.5	90.3	66.0	45.3	82.8	18.7	61.3	64.9	77.4	79.0	14.2	48.9	34.0	59.7
CodeI/O	3.5	75.9	80.7	91.2	67.4	44.9	83.3	22.4	65.0	70.3	77.9	78.7	14.6	49.1	31.3	60.9
CodeI/O++	3.5	73.1	82.0	91.4	66.9	46.0	83.0	26.6	64.4	70.6	78.4	77.8	16.4	49.4	35.3	61.5

1) CODE I/O(++) shows consistent improvements on various benchmarks.
➔ CODE I/O improved general-purpose reasoning ability.

2) CODE I/O(++) gives better performance than raw code dataset (PyEdu).
➔ Training on less structured data is suboptimal.

Experiments - results

High perf. ⟵ ⟶ Low perf.

1st Stage Dataset	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
Qwen 2.5 Coder 7B																
2nd Stage Only		66.9	70.7	83.4	71.6	41.5	77.2	20.7	61.3	60.0	68.3	70.6	10.9	38.7	26.0	54.8
WI	3.5	66.3	73.5	87.0	71.4	39.1	77.5	18.3	59.1	61.6	68.6	68.7	10.2	42.5	26.0	55.0
WI (Full)	11.6	67.0	75.0	87.0	71.1	42.9	78.6	19.1	59.3	59.8	68.4	70.4	10.9	41.9	27.6	55.6
OMI2	3.5	67.6	74.3	84.1	72.3	36.2	77.4	20.9	60.4	61.5	68.8	69.3	10.1	42.7	27.2	55.2
OMI2 (Full)	14.0	66.9	74.0	88.5	73.2	40.9	77.8	19.9	59.5	62.4	68.3	71.3	11.2	41.2	28.4	56.0
OC-SFT-1	4.2	66.6	75.3	86.7	70.9	37.7	78.0	20.3	60.9	60.1	67.5	67.6	10.8	40.1	27.5	55.0
PyEdu	7.7	66.7	74.8	85.8	71.4	40.9	77.4	19.1	58.9	62.4	67.8	65.7	10.6	39.3	25.8	54.8
CodeI/O	3.5	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
CodeI/O++	3.5	66.9	79.1	85.7	72.1	40.6	77.9	24.2	62.5	67.9	71.0	74.2	10.7	45.7	29.1	57.7
LLaMA 3.1 8B																
2nd Stage Only		71.3	73.1	83.2	49.9	40.6	70.0	4.1	44.5	46.9	65.8	65.6	9.8	39.8	25.7	49.3
WI	3.5	72.1	76.3	82.8	52.8	42.9	69.6	4.1	44.0	44.8	64.5	67.8	10.0	42.7	23.1	49.8
OMI2	3.5	72.2	74.8	86.2	58.9	38.2	70.1	5.8	46.1	46.4	67.4	68.6	9.5	40.3	24.5	50.6
OC-SFT-1	4.2	71.0	71.9	81.8	51.1	38.2	68.4	5.7	43.5	44.9	65.6	67.6	10.5	42.0	24.7	49.1
PyEdu	7.7	70.6	69.6	83.2	49.8	42.4	69.1	5.2	43.1	44.5	64.0	65.6	10.2	42.6	25.7	49.0
CodeI/O	3.5	71.7	73.9	83.6	53.8	43.5	69.0	9.3	50.1	53.3	67.5	65.3	10.4	40.9	24.7	51.2
CodeI/O++	3.5	71.8	75.1	84.0	53.2	40.9	68.4	10.0	50.4	53.1	70.0	70.6	10.5	43.2	28.1	52.1

1st Stage Dataset	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
DeepSeek Coder v2 Lite 16B																
2nd Stage Only		68.4	73.4	82.5	60.0	38.6	68.5	14.8	53.0	54.9	61.1	69.2	6.7	44.7	26.6	51.6
WI	3.5	68.5	73.8	83.7	60.5	39.5	68.7	14.3	53.5	57.1	61.6	65.7	6.9	43.1	25.4	51.6
OMI2	3.5	67.6	74.1	84.7	64.7	38.4	70.1	14.4	53.8	55.8	63.6	66.4	6.4	42.0	24.7	51.9
OC-SFT-1	4.2	68.2	73.6	83.3	60.9	37.3	69.1	14.7	52.8	56.1	60.9	67.9	6.1	42.7	25.2	51.3
PyEdu	7.7	68.3	74.6	83.0	60.6	38.2	69.7	15.6	54.9	57.0	61.9	68.6	7.0	44.7	24.6	52.1
CodeI/O	3.5	68.4	74.6	83.6	60.9	38.6	70.3	18.7	58.4	62.8	63.1	70.8	7.8	46.0	26.1	53.6
CodeI/O++	3.5	69.0	73.5	82.8	60.9	38.8	70.0	20.3	59.5	61.0	64.2	69.4	6.7	46.3	26.9	53.5
Gemma 2 27B																
2nd Stage Only		72.4	80.1	90.1	66.3	44.4	82.8	19.1	62.5	66.9	77.1	80.4	13.5	47.8	30.0	59.5
WI	3.5	73.2	79.0	91.5	70.6	44.9	82.7	20.7	63.5	66.3	77.6	77.2	17.1	47.3	33.3	60.4
OMI2	3.5	73.1	79.3	90.8	67.1	44.0	83.4	19.2	61.4	66.0	77.1	80.5	13.9	49.7	40.7	60.4
OC-SFT-1	4.2	73.5	79.9	91.1	66.1	46.9	81.8	20.2	62.8	65.6	77.3	78.9	14.0	46.9	35.3	60.0
PyEdu	7.7	73.7	79.5	90.3	66.0	45.3	82.8	18.7	61.3	64.9	77.4	79.0	14.2	48.9	34.0	59.7
CodeI/O	3.5	75.9	80.7	91.2	67.4	44.9	83.3	22.4	65.0	70.3	77.9	78.7	14.6	49.1	31.3	60.9
CodeI/O++	3.5	73.1	82.0	91.4	66.9	46.0	83.0	26.6	64.4	70.6	78.4	77.8	16.4	49.4	35.3	61.5

3) CODE I/O ++ outperforms CODE I/O.

➔ Multi-turn revision improved dataset quality and reasoning ability of the model.

Analysis

Ablation Study : Input/Output Prediction

- Input and output prediction by training on each separately
  - I. Pred. only : training only with input prediction data
  - O. Pred. only : training only with output prediction data
- The scores are generally similar

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of prediction inputs or outputs only.																
I. Pred. only	1.75	66.3	75.9	85.8	71.6	38.8	77.7	22.9	62.8	64.5	68.3	69.4	11.4	44.4	26.2	56.1
O. Pred. only	1.76	66.9	75.2	84.6	71.5	42.4	76.5	23.3	61.1	65.6	70.1	72.1	11.4	42.2	26.9	56.4

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Input/Output Prediction

- Input and output prediction by training on each separately
  - I. Pred. only : training only with input prediction data
  - O. Pred. only : training only with output prediction data
- The scores are generally similar

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of prediction inputs or outputs only.																
I. Pred. only	1.75	66.3	75.9	85.8	71.6	38.8	77.7	22.9	62.8	64.5	68.3	69.4	11.4	44.4	26.2	56.1
O. Pred. only	1.76	66.9	75.2	84.6	71.5	42.4	76.5	23.3	61.1	65.6	70.1	72.1	11.4	42.2	26.9	56.4

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Input/Output Prediction

- Input and output prediction by training on each separately
  - I. Pred. only : training only with input prediction data
  - O. Pred. only : training only with output prediction data
- **Incorporating both input-output predictions can lead to more balanced performance !**

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of prediction inputs or outputs only.																
I. Pred. only	1.75	66.3	75.9	85.8	71.6	38.8	77.7	22.9	62.8	64.5	68.3	69.4	11.4	44.4	26.2	56.1
O. Pred. only	1.76	66.9	75.2	84.6	71.5	42.4	76.5	23.3	61.1	65.6	70.1	72.1	11.4	42.2	26.9	56.4

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Rejection Sampling

- Rejection sampling : Filtering incorrect responses

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of rejection sampling.																
w/o wrong	1.79	66.8	74.9	87.4	71.5	39.1	76.7	22.6	62.6	66.6	68.3	71.9	11.5	42.6	27.8	56.5
wrong→gt	3.52	66.4	76.8	86.0	70.6	42.4	76.5	24.3	62.1	67.6	68.0	71.1	11.5	43.1	26.6	56.6

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Rejection Sampling

- Rejection sampling : Filtering incorrect responses

  → Remove 50% of the training data (w/o wrong)

- This results in a general performance drop, suggesting a loss of data diversity

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of rejection sampling.																
w/o wrong	1.79	66.8	74.9	87.4	71.5	39.1	76.7	22.6	62.6	66.6	68.3	71.9	11.5	42.6	27.8	56.5
wrong→gt	3.52	66.4	76.8	86.0	70.6	42.4	76.5	24.3	62.1	67.6	68.0	71.1	11.5	43.1	26.6	56.6

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Rejection Sampling

- Rejection sampling : Filtering incorrect responses
  → Remove 50% of the training data
- There is no significant benefit when using a subset dataset that only uses 50% of the CodeI/O (~50% subset)

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of rejection sampling.																
w/o wrong	1.79	66.8	74.9	87.4	71.5	39.1	76.7	22.6	62.6	66.6	68.3	71.9	11.5	42.6	27.8	56.5
wrong→gt	3.52	66.4	76.8	86.0	70.6	42.4	76.5	24.3	62.1	67.6	68.0	71.1	11.5	43.1	26.6	56.6

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Rejection Sampling

- wrong → gt : Replacing all incorrect responses with ground-truth answers through code execution
- It still has no advantages

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of rejection sampling.																
w/o wrong	1.79	66.8	74.9	87.4	71.5	39.1	76.7	22.6	62.6	66.6	68.3	71.9	11.5	42.6	27.8	56.5
wrong→gt	3.52	66.4	76.8	86.0	70.6	42.4	76.5	24.3	62.1	67.6	68.0	71.1	11.5	43.1	26.6	56.6

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Ablation Study : Rejection Sampling

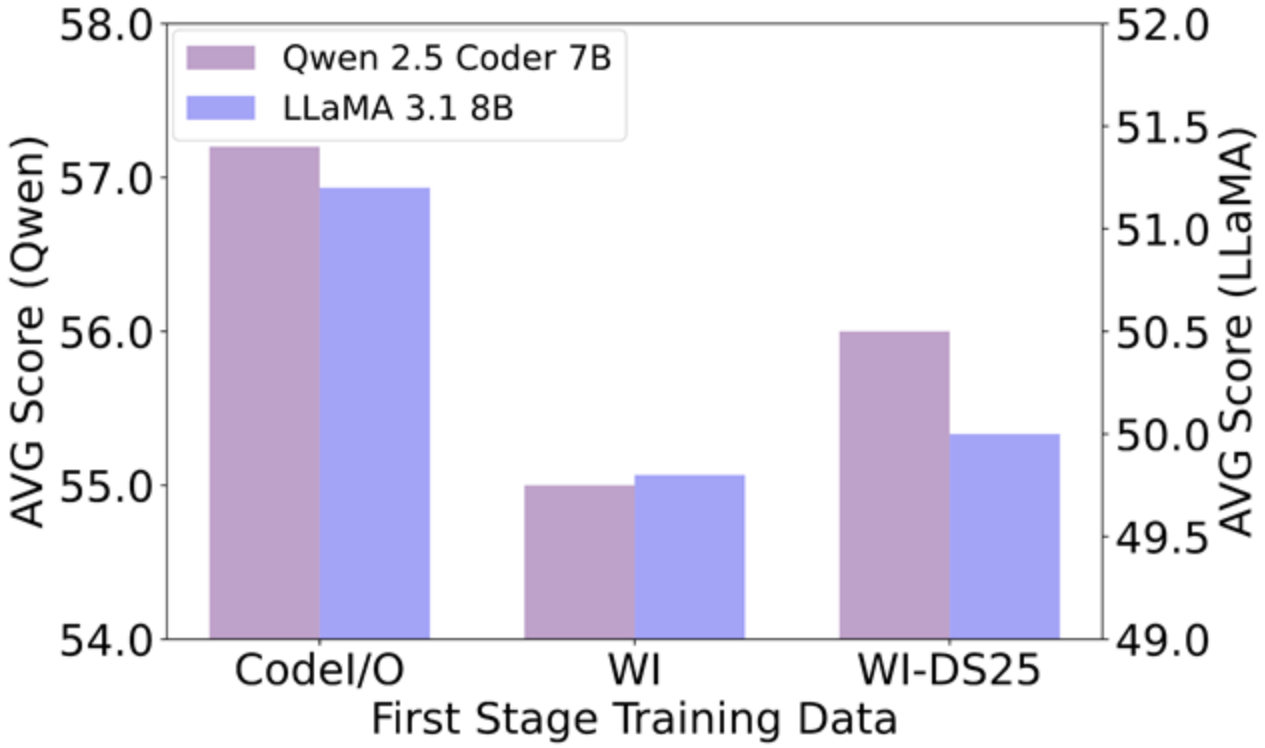
- wrong → gt : Replacing all incorrect responses with ground-truth answers through code execution
- It still has no advantages
- **To maintain performance balance, we retain all incorrect responses !**

	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
CODEI/O	3.52	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
~ 50% subset	1.59	67.5	74.7	86.7	71.6	42.9	77.3	23.0	62.8	65.9	69.1	70.8	10.5	42.1	28.9	56.7
Effect of rejection sampling.																
w/o wrong	1.79	66.8	74.9	87.4	71.5	39.1	76.7	22.6	62.6	66.6	68.3	71.9	11.5	42.6	27.8	56.5
wrong→gt	3.52	66.4	76.8	86.0	70.6	42.4	76.5	24.3	62.1	67.6	68.0	71.1	11.5	43.1	26.6	56.6

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Effect of Different Synthesis Model

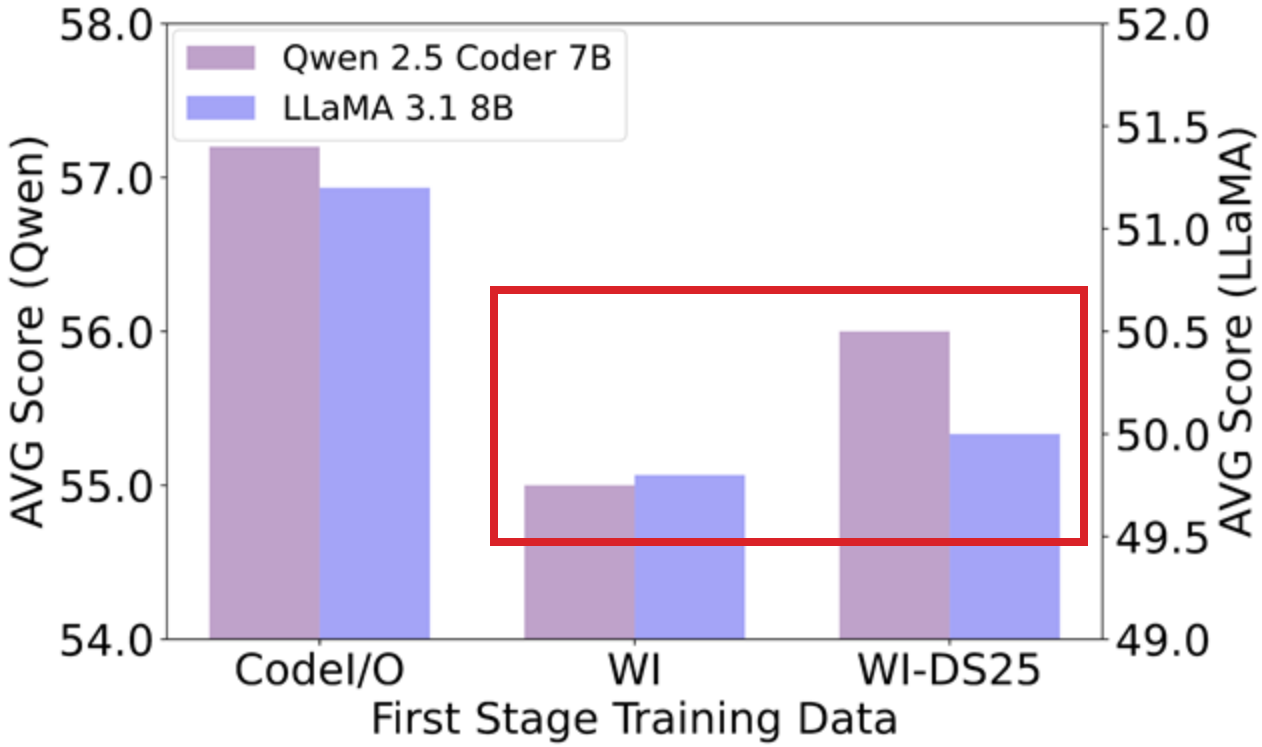
- WebInstruct (WI) : Dataset refined by Qwen-72B or Mixtral-22Bx8
- WI-DS25 : Dataset refined by DeepSeek-V2.5



[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.
[2] Yue, Xiang, et al. "Mammoth2: Scaling instructions from the web." NeurIPS, 2024.

Effect of Different Synthesis Model

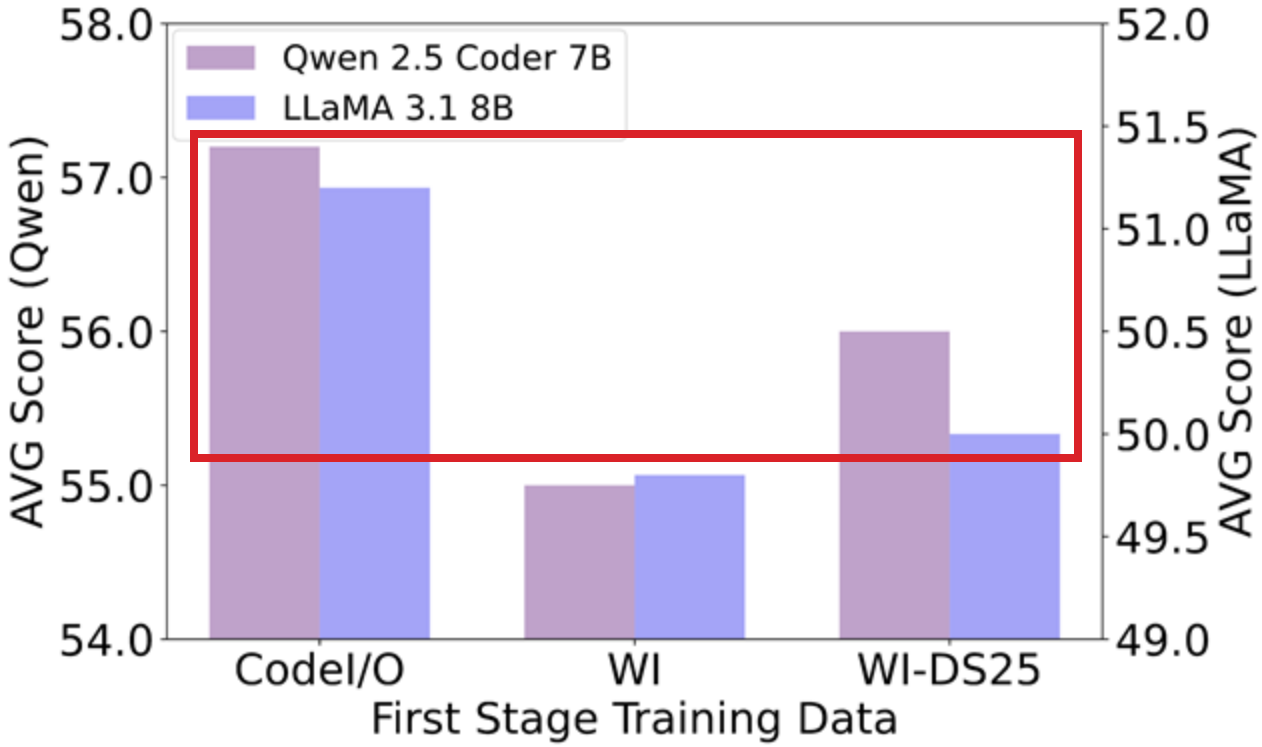
- WebInstruct (WI) : Dataset refined by Qwen-72B or Mixtral-22Bx8
- WI-DS25 : Dataset refined by DeepSeek-V2.5



[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.
[2] Yue, Xiang, et al. "Mammoth2: Scaling instructions from the web." NeurIPS, 2024.

Effect of Different Synthesis Model

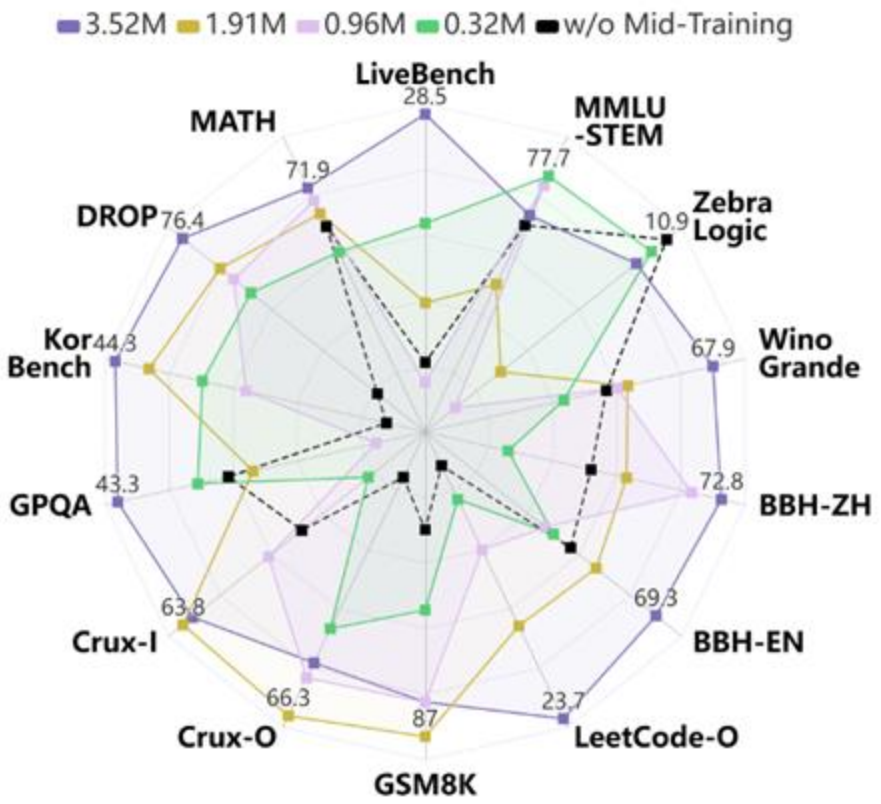
- WebInstruct (WI) : Dataset refined by Qwen-72B or Mixtral-22Bx8
- WI-DS25 : Dataset refined by DeepSeek-V2.5



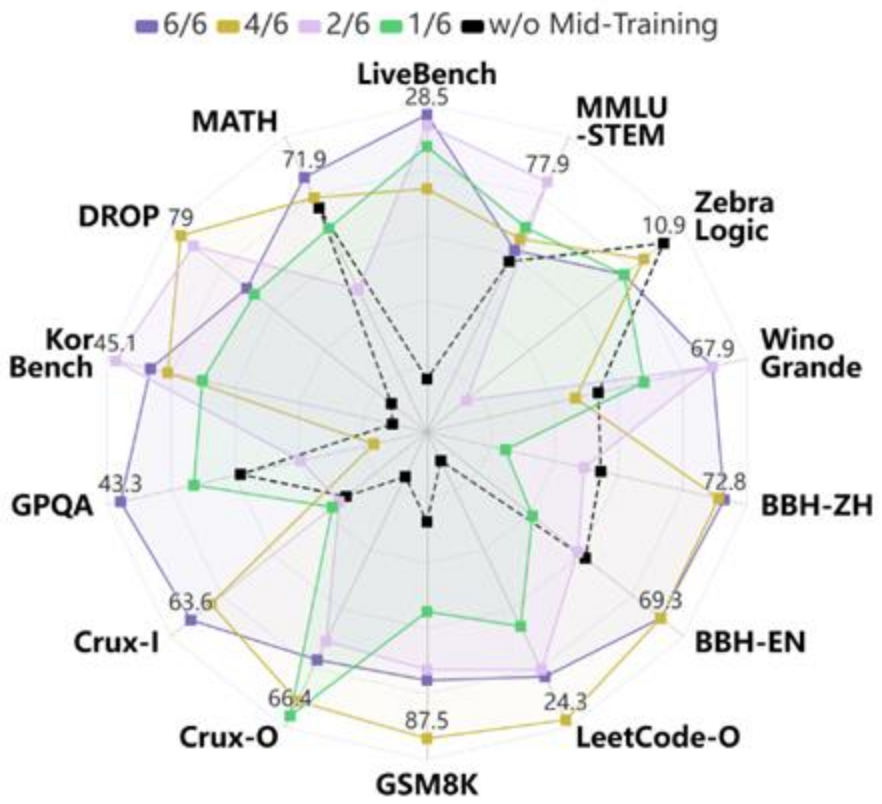
[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.
[2] Yue, Xiang, et al. "Mammoth2: Scaling instructions from the web." NeurIPS, 2024.

Scaling Effect of CodeI/O

- How CodeI/O scales with varying amounts of training data



(a) Size of randomly sampled subset.

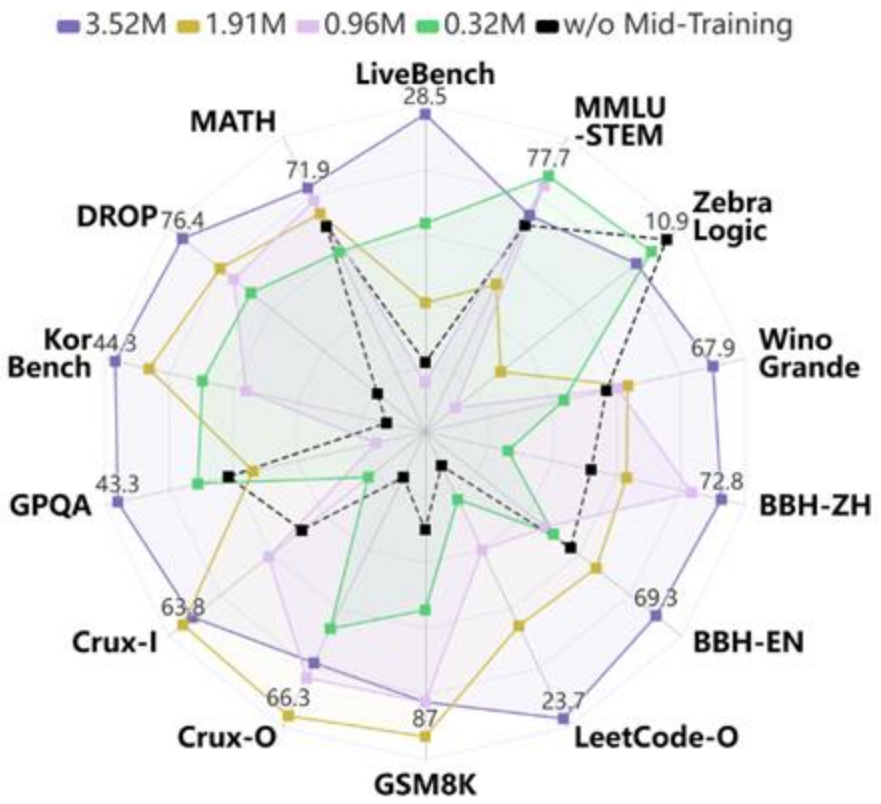


(b) Ratio of testcases per sample compared to the full set.

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Scaling Effect of CodeI/O

- How CodeI/O scales with varying amounts of training data
- Just increasing the number of training samples → **Clear benefit !**



(a) Size of randomly sampled subset.

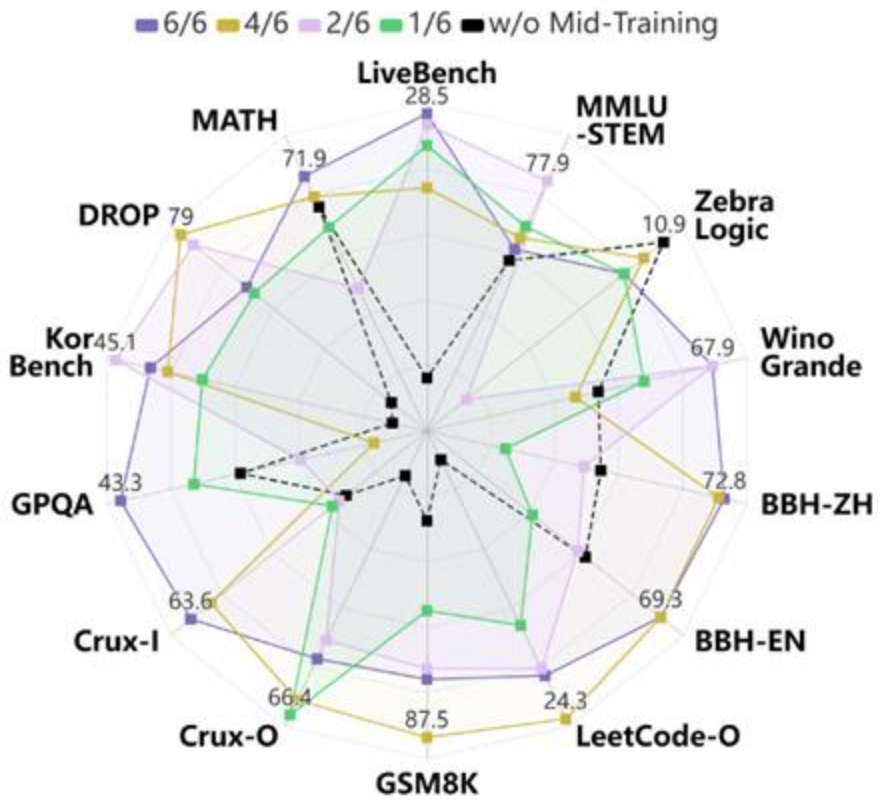
(b) Ratio of testcases per sample compared to the full set.

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Scaling Effect of CodeI/O

- How CodeI/O scales with varying amounts of training data
- Increasing the ratio of input-output pairs → **Generally benefit !**

(a) Size of randomly sampled subset.



(b) Ratio of testcases per sample compared to the full set.

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Different Data Format

- How to best arrange the query, reference code, CoT in training samples

Data Format		Wino		GSM			MMLU	LC	CRUX		BBH		Zebra	Kor	Live	
Prompt	Response	Grande	DROP	8K	MATH	GPQA	-STEM	-O	-I	-O	-EN	-ZH	Logic	Bench	Bench	AVG
Q+Code	CoT	**67.9**	76.4	86.4	**71.9**	**43.3**	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	**28.5**	**57.2**
Q	CoT	67.2	**76.8**	87.2	70.4	37.5	77.3	**25.2**	62.6	65.3	69.2	71.1	**11.5**	44.9	**28.5**	56.8
Code	CoT	**67.9**	76.4	87.0	70.8	39.5	76.5	25.0	**64.1**	65.8	68.8	71.3	10.6	**45.2**	**28.5**	57.0
Q	Code+CoT	65.9	76.1	**87.5**	71.7	42.2	76.9	22.9	63.9	**66.1**	**69.6**	**72.9**	10.9	41.4	**28.5**	56.9
Q	Code	66.9	73.1	84.8	71.6	40.0	**77.4**	20.8	59.5	62.4	67.2	68.3	10.1	40.3	26.3	54.9

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Different Data Format

- How to best arrange the query, reference code, CoT in training samples
- Prompt (Q+Code) + Response (CoT) → Best performance !

Data Format		Wino		GSM			MMLU	LC	CRUX		BBH		Zebra	Kor	Live	
Prompt	Response	Grande	DROP	8K	MATH	GPQA	-STEM	-O	-I	-O	-EN	-ZH	Logic	Bench	Bench	AVG
Q+Code	CoT	**67.9**	76.4	86.4	**71.9**	**43.3**	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	**28.5**	**57.2**
Q	CoT	67.2	**76.8**	87.2	70.4	37.5	77.3	**25.2**	62.6	65.3	69.2	71.1	**11.5**	44.9	**28.5**	56.8
Code	CoT	**67.9**	76.4	87.0	70.8	39.5	76.5	25.0	**64.1**	65.8	68.8	71.3	10.6	**45.2**	**28.5**	57.0
Q	Code+CoT	65.9	76.1	**87.5**	71.7	42.2	76.9	22.9	63.9	**66.1**	**69.6**	**72.9**	10.9	41.4	**28.5**	56.9
Q	Code	66.9	73.1	84.8	71.6	40.0	**77.4**	20.8	59.5	62.4	67.2	68.3	10.1	40.3	26.3	54.9

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Different Data Format

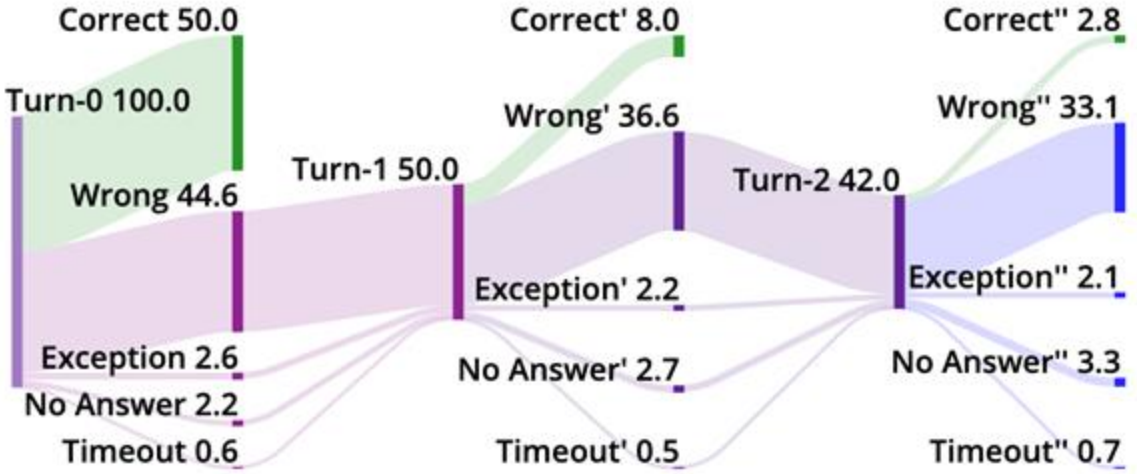
- How to best arrange the query, reference code, CoT in training samples
- Prompt (Q+Code) + Response (CoT) → Best performance !
- **Without CoT, performance drops**

Data Format		Wino		GSM			MMLU	LC	CRUX		BBH		Zebra	Kor	Live	
Prompt	Response	Grande	DROP	8K	MATH	GPQA	-STEM	-O	-I	-O	-EN	-ZH	Logic	Bench	Bench	AVG
Q+Code	CoT	**67.9**	76.4	86.4	**71.9**	**43.3**	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	**28.5**	**57.2**
Q	CoT	67.2	**76.8**	87.2	70.4	37.5	77.3	**25.2**	62.6	65.3	69.2	71.1	**11.5**	44.9	**28.5**	56.8
Code	CoT	**67.9**	76.4	87.0	70.8	39.5	76.5	25.0	**64.1**	65.8	68.8	71.3	10.6	**45.2**	**28.5**	57.0
Q	Code+CoT	65.9	76.1	**87.5**	71.7	42.2	76.9	22.9	63.9	**66.1**	**69.6**	**72.9**	10.9	41.4	**28.5**	56.9
Q	Code	66.9	73.1	84.8	71.6	40.0	**77.4**	20.8	59.5	62.4	67.2	68.3	10.1	40.3	26.3	54.9

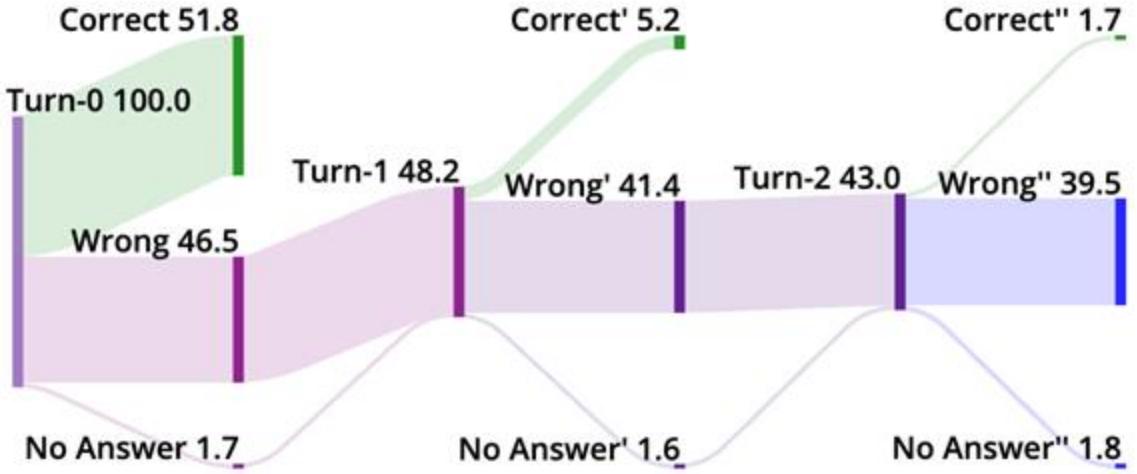
[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Multi-turn Revision

- Extension revision to a second turn to evaluate further improvements
- Most correct responses are predicted in the initial turn
- About 10% of incorrect responses corrected in the first-turn revision



(a) Input prediction revision.

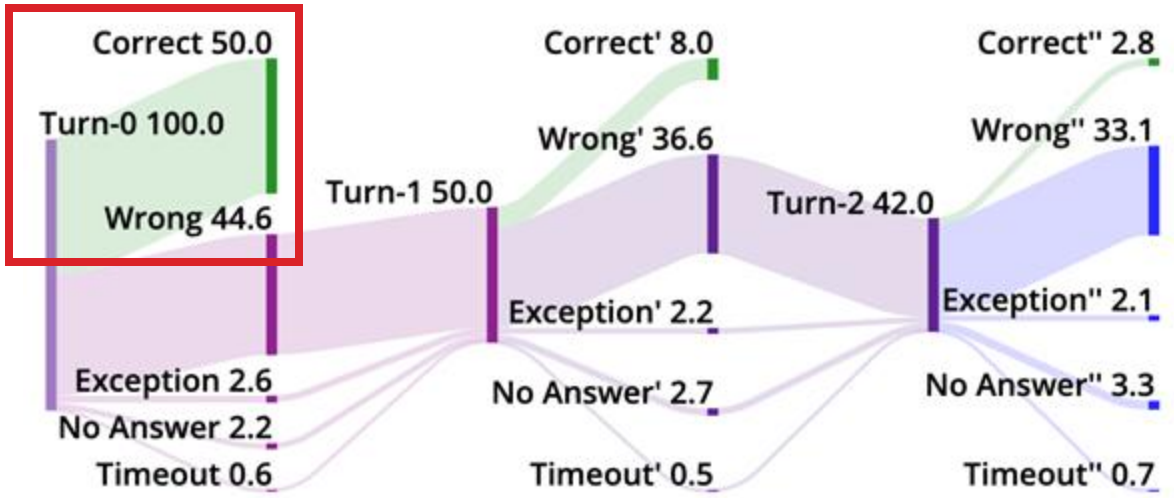


(b) Output prediction revision.

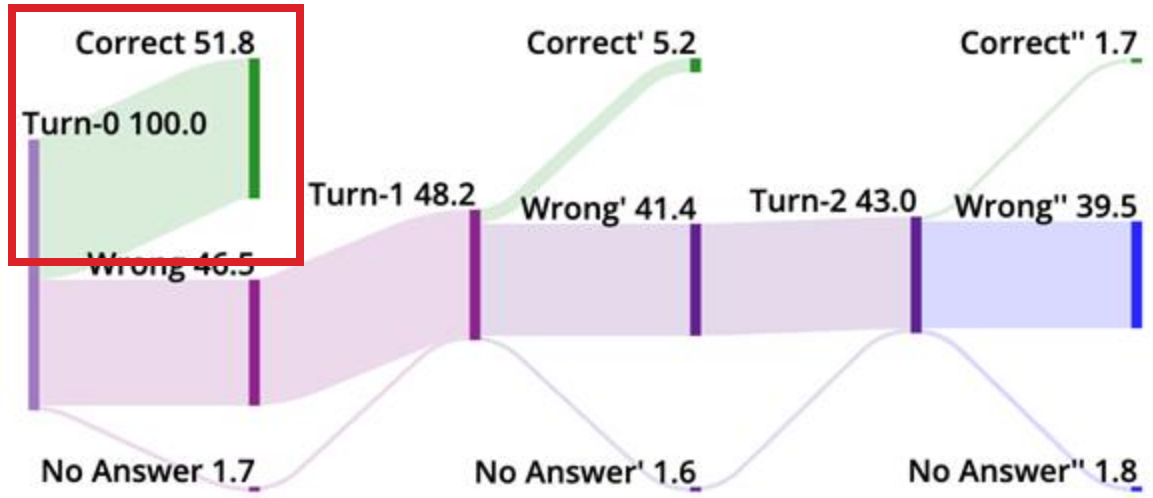
[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Multi-turn Revision

- Extension revision to a second turn to evaluate further improvements
- Most correct responses are predicted in the initial turn
- About 10% of incorrect responses corrected in the first-turn revision



(a) Input prediction revision.

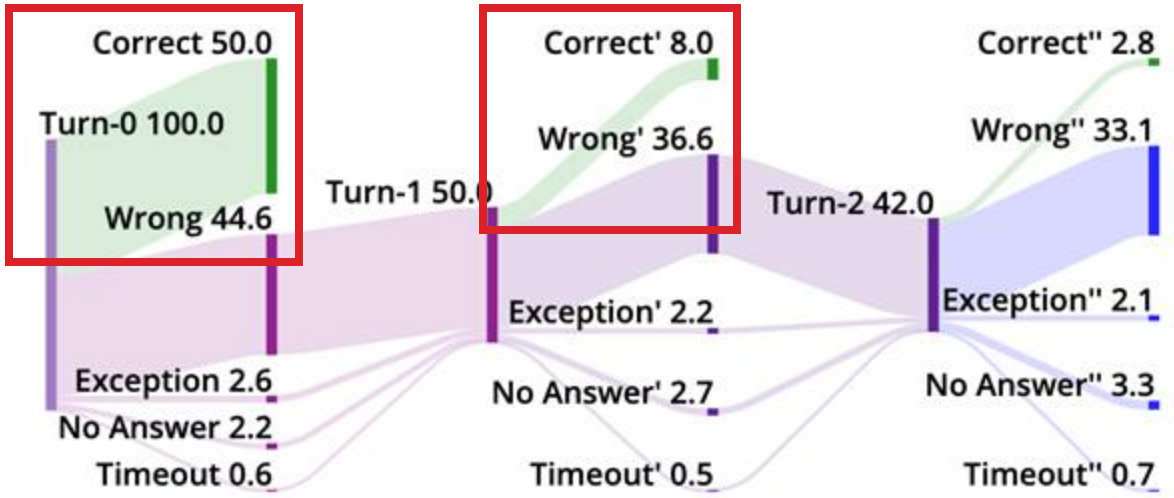


(b) Output prediction revision.

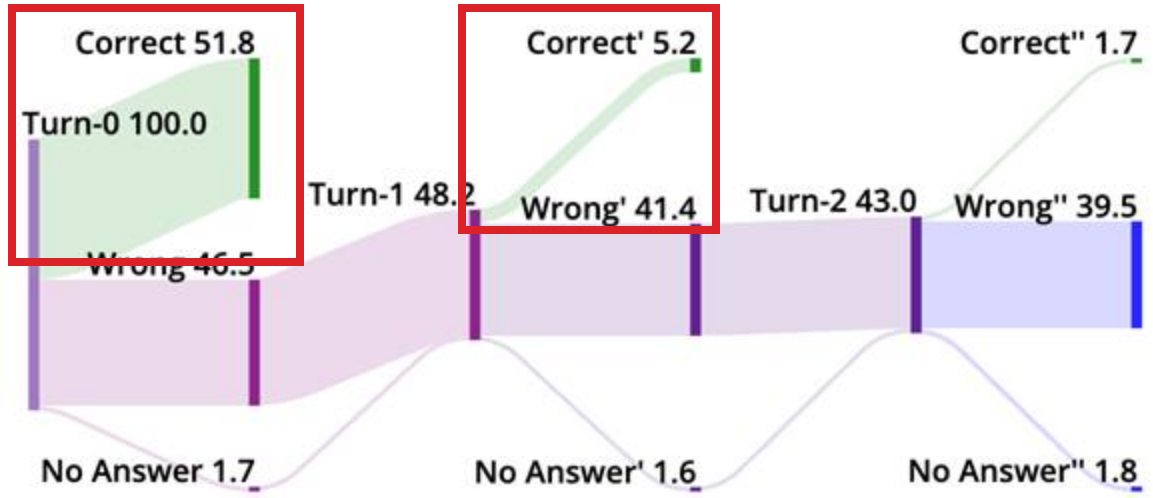
[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Multi-turn Revision

- Extension revision to a second turn to evaluate further improvements
- Most correct responses are predicted in the initial turn
- About 10% of incorrect responses corrected in the first-turn revision



(a) Input prediction revision.

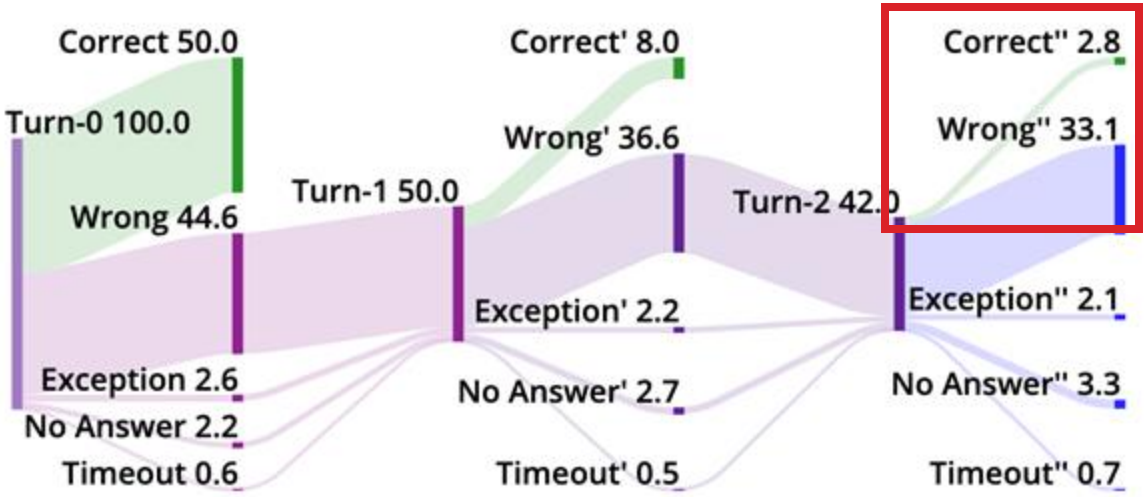


(b) Output prediction revision.

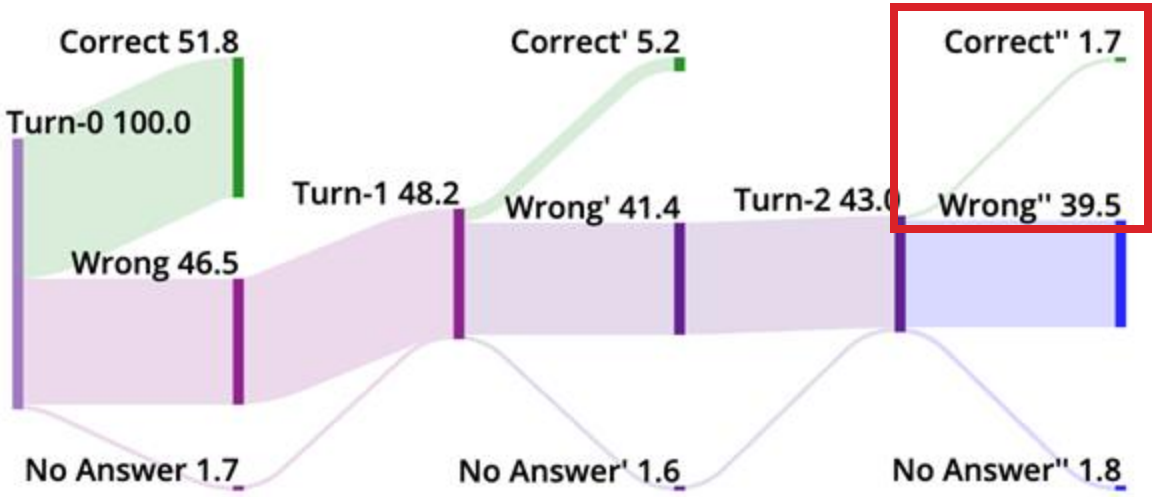
[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Multi-turn Revision

- However, the second turn yields significantly fewer corrections
- Minimal gains or even performance drops



(a) Input prediction revision.

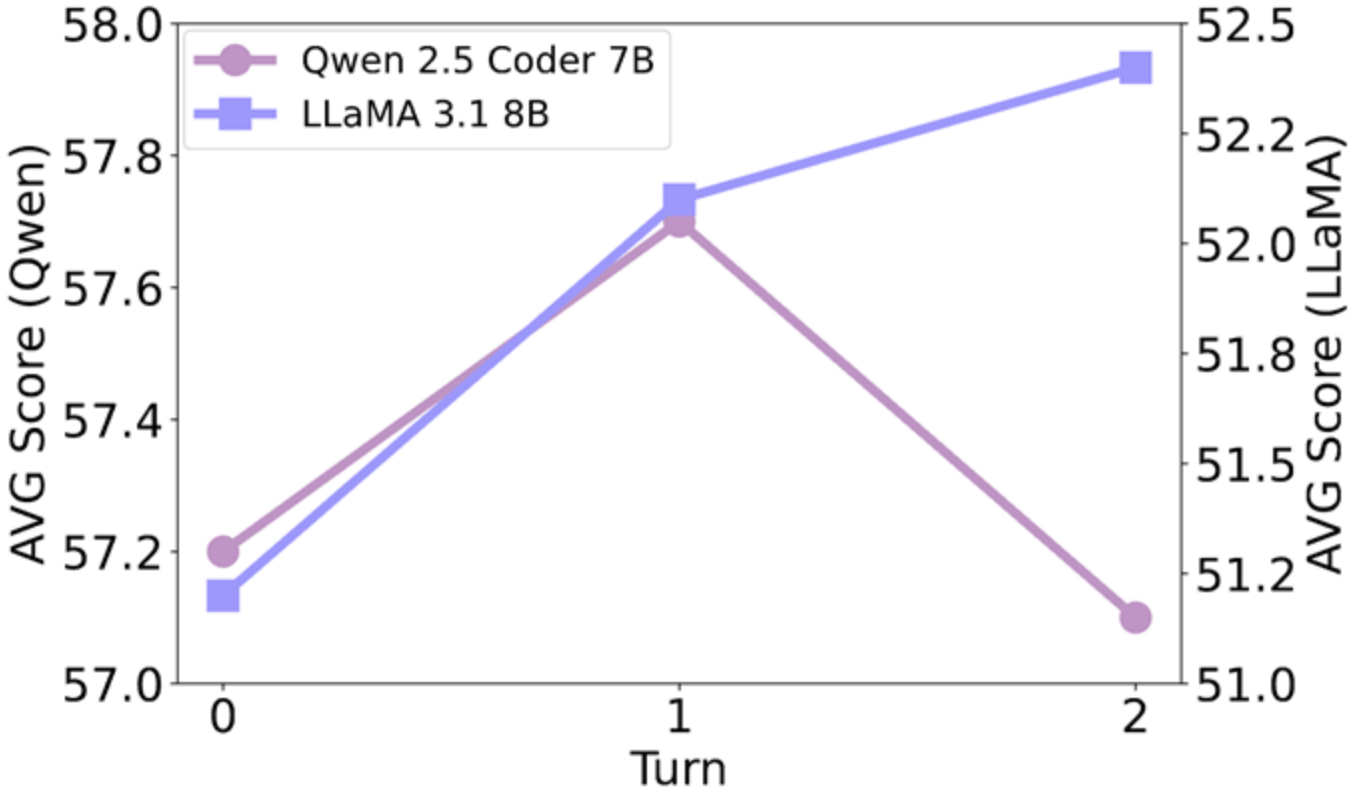


(b) Output prediction revision.

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Multi-turn Revision

- However, the second turn yields significantly fewer corrections
- Minimal gains or even performance drops → Using single-turn revision !



[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

The Necessity of Two Stage Training

- Experiments about the necessity of a separate training stage with CodeI/O data
- All two-stage variants outperform single-stage training

First Stage	Second Stage	Model Qwen	Model LLaMA
-	IT	54.8	49.3
-	CODEI/O(10%)+IT	56.6	50.5
CODEI/O+IT	-	55.9	49.7
CODEI/O	IT	57.2	51.2
CODEI/O+IT	IT	56.8	51.5
CODEI/O	CODEI/O(10%)+IT	57.0	52.7

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

The Necessity of Two Stage Training

- Experiments about the necessity of a separate training stage with CodeI/O data
- All two-stage variants outperform single-stage training

First Stage	Second Stage	Model Qwen	Model LLaMA
-	IT	54.8	49.3
-	CODEI/O(10%)+IT	56.6	50.5
CODEI/O+IT	-	55.9	49.7
CODEI/O	IT	57.2	51.2
CODEI/O+IT	IT	56.8	51.5
CODEI/O	CODEI/O(10%)+IT	57.0	52.7

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Analysis Summary

- To maintain performance balance, we retain all incorrect responses and incorporate both input-output predictions, not knowledge distillation from an advanced model

- Scaling the number of samples or the ratio of I/O helps capturing and learning complex logic flow

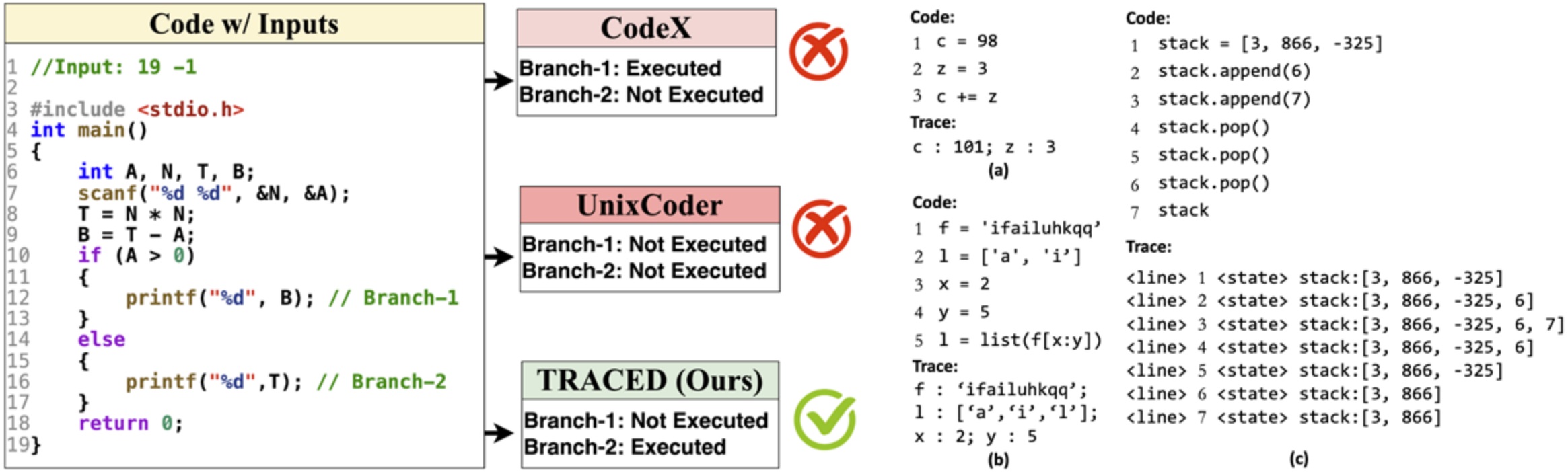
- Including CoT and two-stage training lead higher performance

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Related Work

Learning about Code Execution

- Most related work focus solely on the code execution output prediction task itself
- Other works also utilize code execution to improve code generation abilities



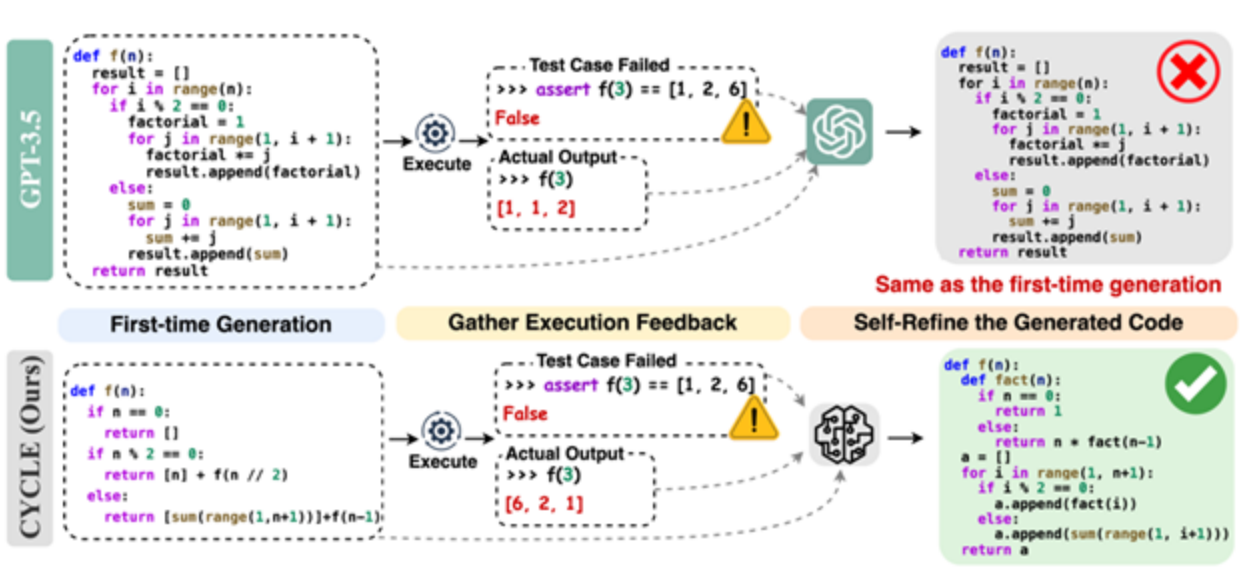
[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

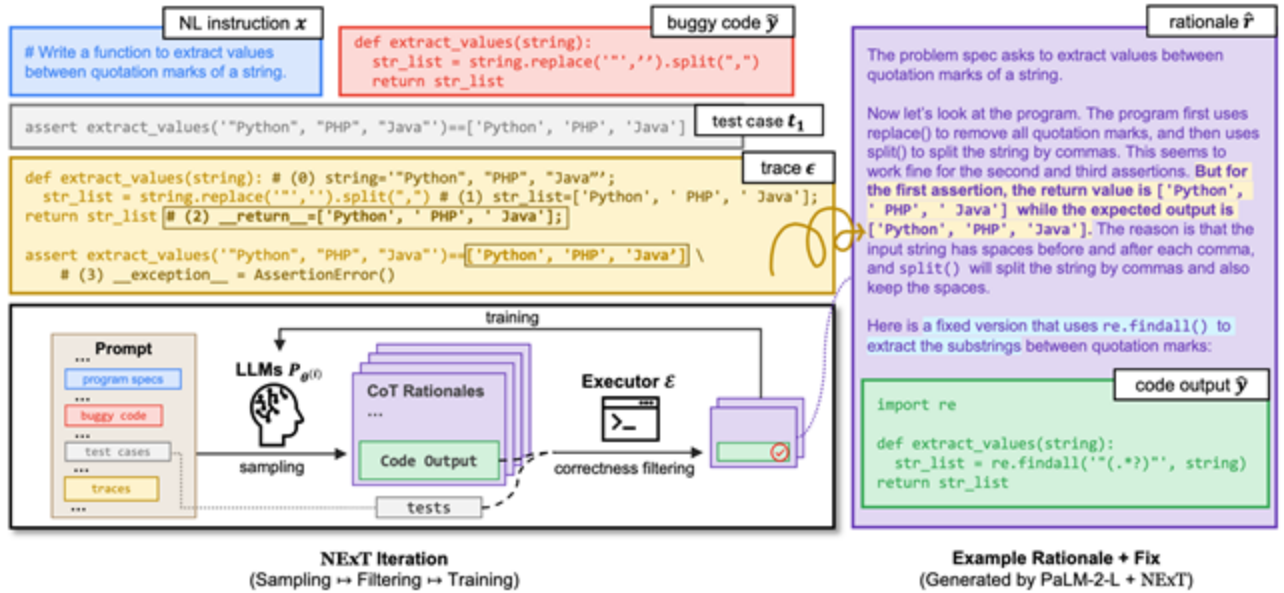
[2] Liu, Chenxiao, et al. "Code execution with pre-trained language models." arXiv, 2023.

[3] Ding, Yangruibo, et al. "Traced: Execution-aware pre-training for source code." Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, 2024.

Learning about Code Execution

- Most related work focus solely on the code execution output prediction task itself
- Other works also utilize code execution to improve code generation abilities





[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

[2] Ding, Yangruibo, et al. "Cycle: Learning to self-refine the code generation." Proceedings of the ACM on Programming Languages, 2024.

[3] Ni, Ansong, et al. "Next: Teaching large language models to reason about code execution." arXiv, 2024.

Learning about Code Execution

- Most related work focus solely on the code execution output prediction task itself
- Other works also utilize code execution to improve code generation abilities

→ Predict diverse code input-output and demonstrate efficacy in general reasoning abilities, not code-specific task !

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.
[2] Ding, Yangruibo, et al. "Cycle: Learning to self-refine the code generation." Proceedings of the ACM on Programming Languages, 2024.
[3] Ni, Ansong, et al. "Next: Teaching large language models to reason about code execution." arXiv, 2024.

Inference Time Scaling

- Inference-time scaling : Encouraging models to generate ultra-long reasoning process to solve problems through large-scale reinforcement learning

[1] Li, Junlong, et al. "Codel/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.
[2] Jaech, Aaron, et al. "Openai o1 system card." arXiv, 2024.
[3] Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." arXiv, 2025.

Inference Time Scaling

- Inference-time scaling : Encouraging models to generate ultra-long reasoning process to solve problems through large-scale reinforcement learning

⇒ CodeI/O is orthogonal to this method and can provide a better basis to further incentivize the reasoning abilities of LLMs

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.
[2] Jaech, Aaron, et al. "Openai o1 system card." arXiv, 2024.
[3] Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." arXiv, 2025.

Conclusion

Conclusion

- Introducing CodeI/O to improve the reasoning abilities of LLMs by training them to predict code inputs and outputs in pure CoTs
- This approach can enhance general reasoning abilities, including symbolic, logical, mathematical, and commonsense reasoning

[1] Li, Junlong, et al. "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction." arXiv, 2025.

Limitation

- Inherent limitations in the diversity and accuracy of the DeepSeek
- Codel/O++ is introduced as an advanced dataset with multi-turn revision, but experiment shows that multi-turn revision is not significant
- No such verification when generating labels for input prediction and output prediction stages
- Evaluation problem : Multiple valid inputs can produce the same output → Correct input prediction is labeled as wrong prediction
- Despite claims of general reasoning, the model's performance is insufficient and needs deeper evaluation