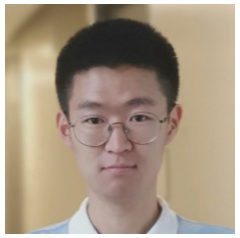
Reasoning Runtime Behavior of a Program with LLM: How Far Are We?

Junkai Chen*[1]

Zhiyuan Pan*[1]

Xing Hu[1]



Zhenhao Li[2]

Ge Li[3]

Xin Xia[1]

* Equal Contribution

[1] Zhejiang University, [2] York University, [3] Peking University

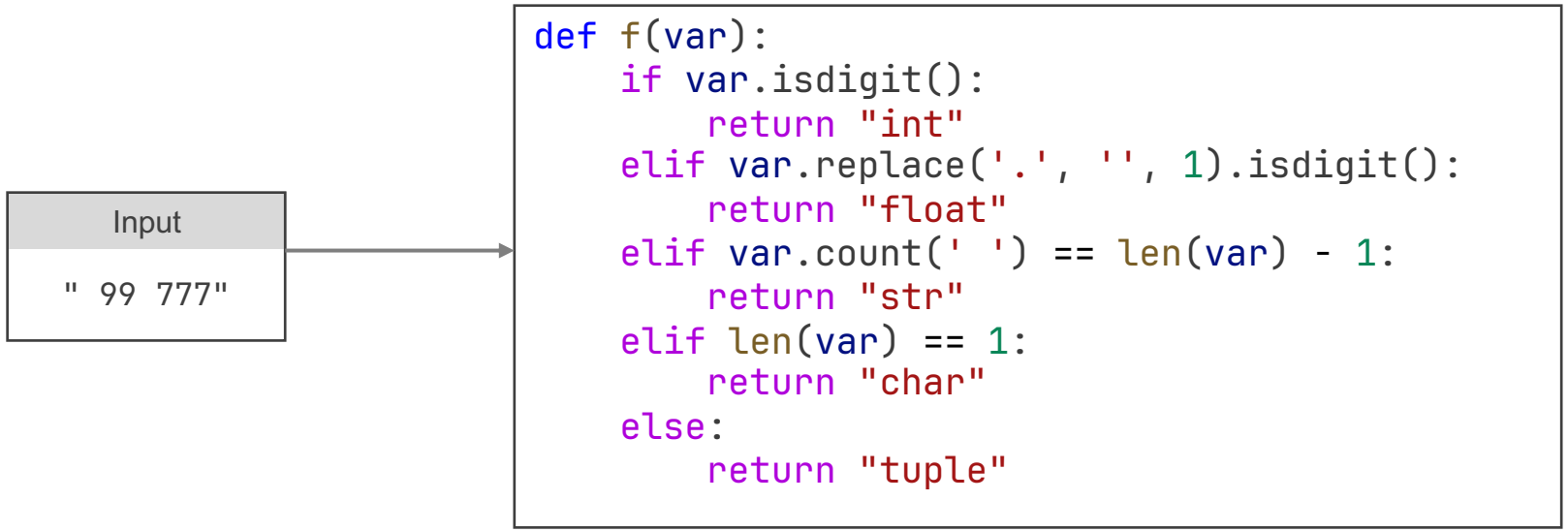
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*

```python
def f(var):
    if var.isdigit():
        return "int"
    elif var.replace('.', '', 1).isdigit():
        return "float"
    elif var.count(' ') == len(var) - 1:
        return "str"
    elif len(var) == 1:
        return "char"
    else:
        return "tuple"
```

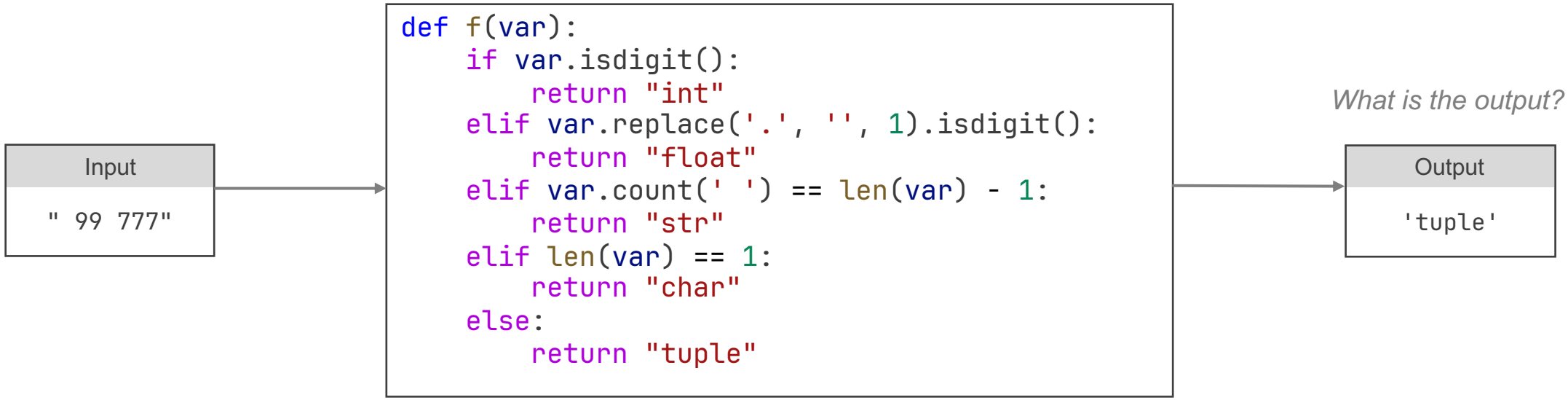
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



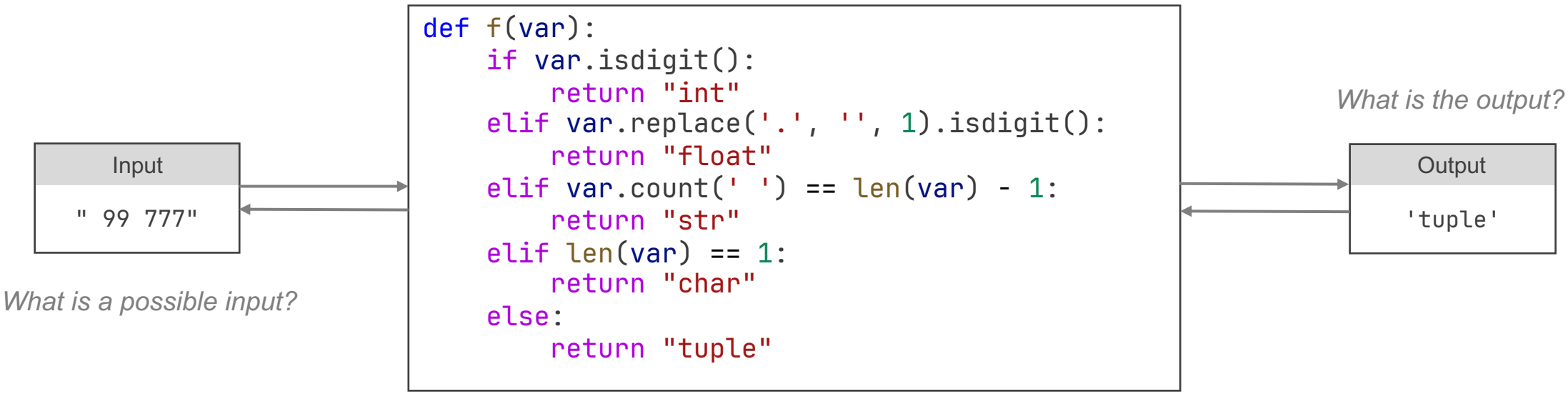
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



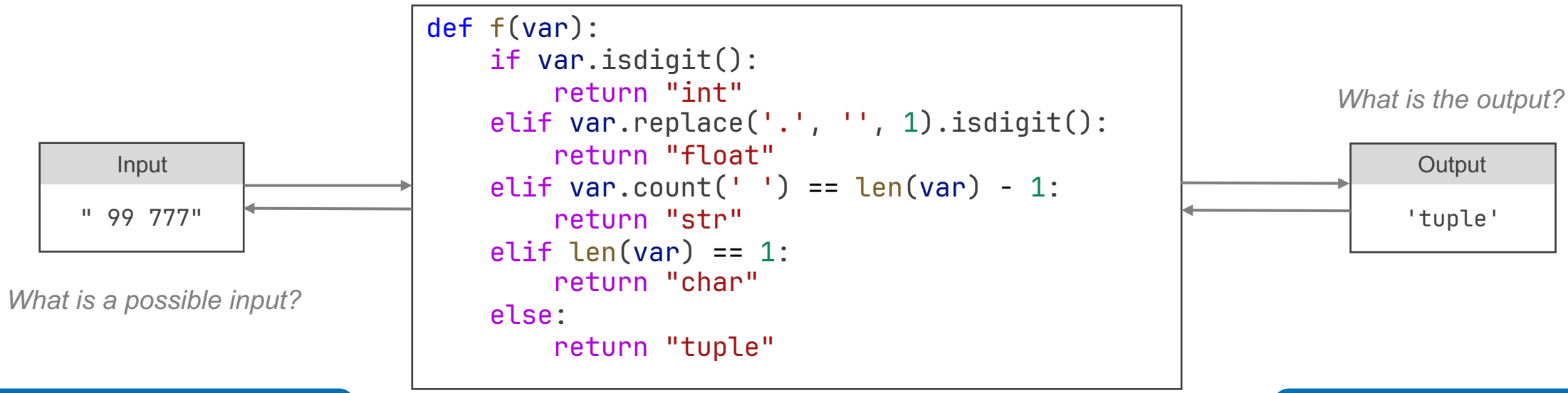
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*

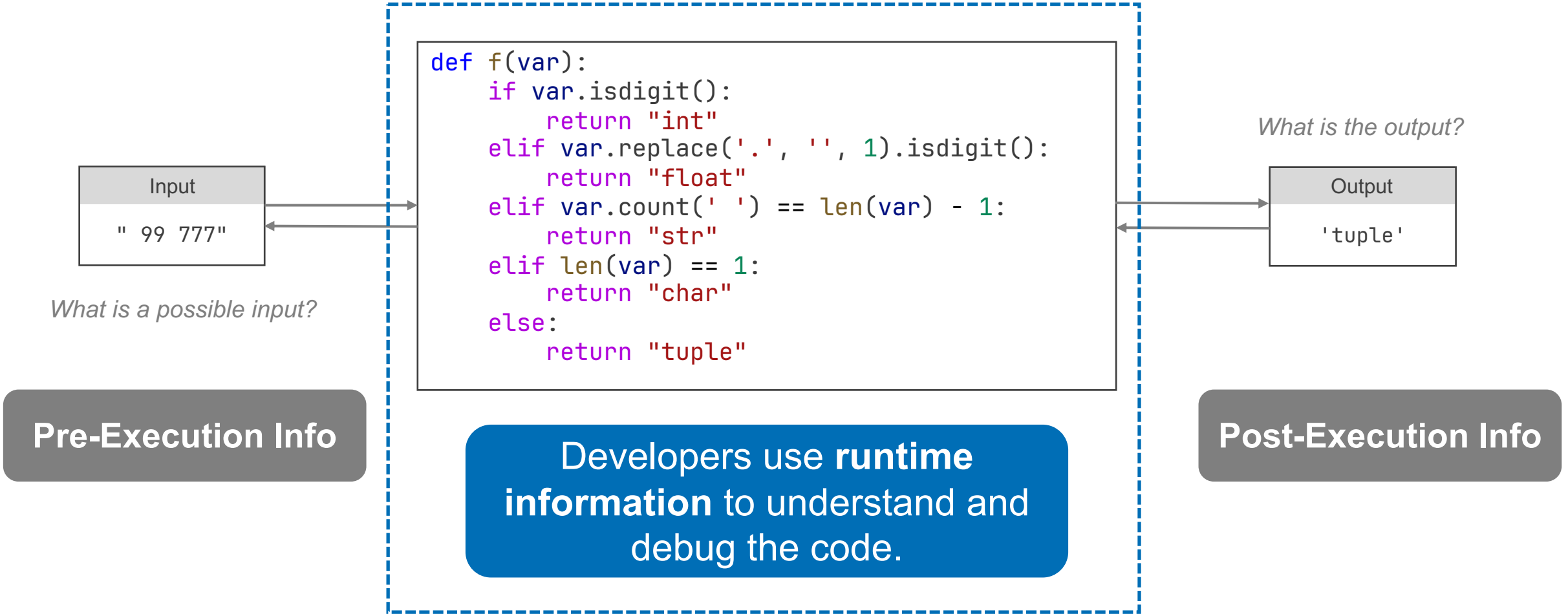


Previous benchmark[1]: Pre-/Post-Execution Information

1. Gu et al. "CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution." *ICML 2024*.

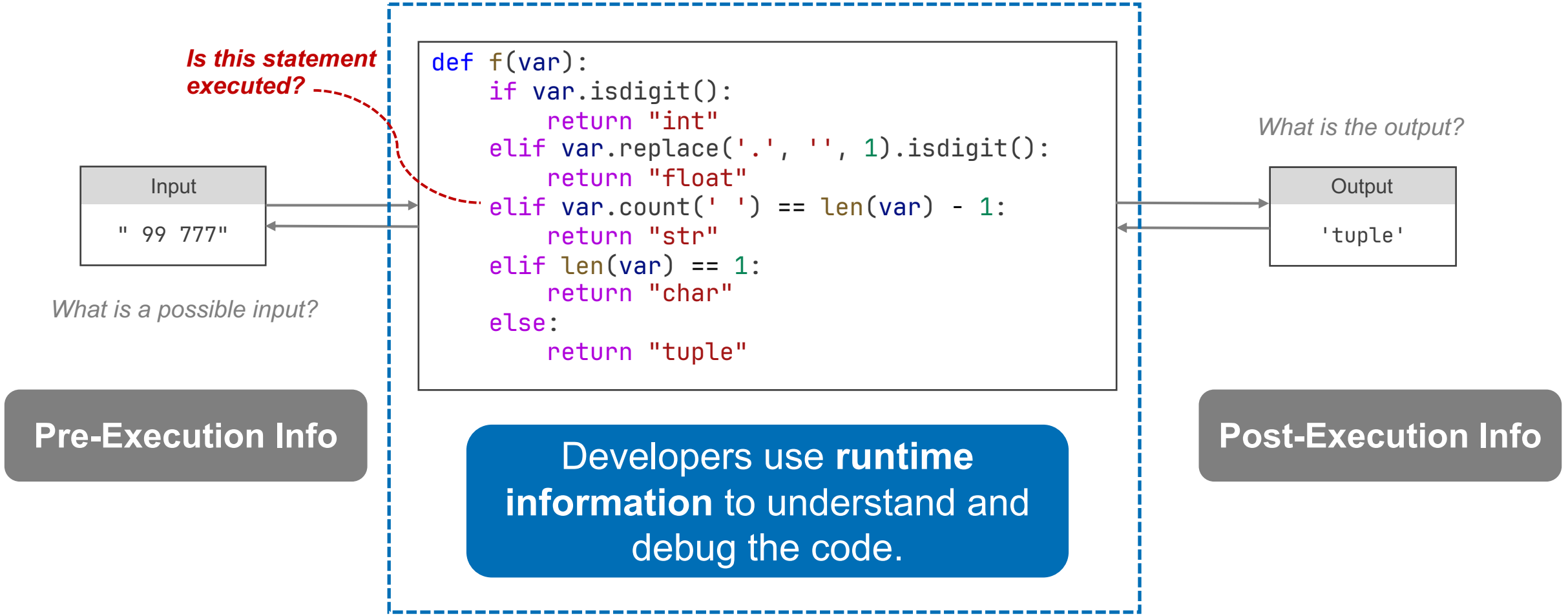
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



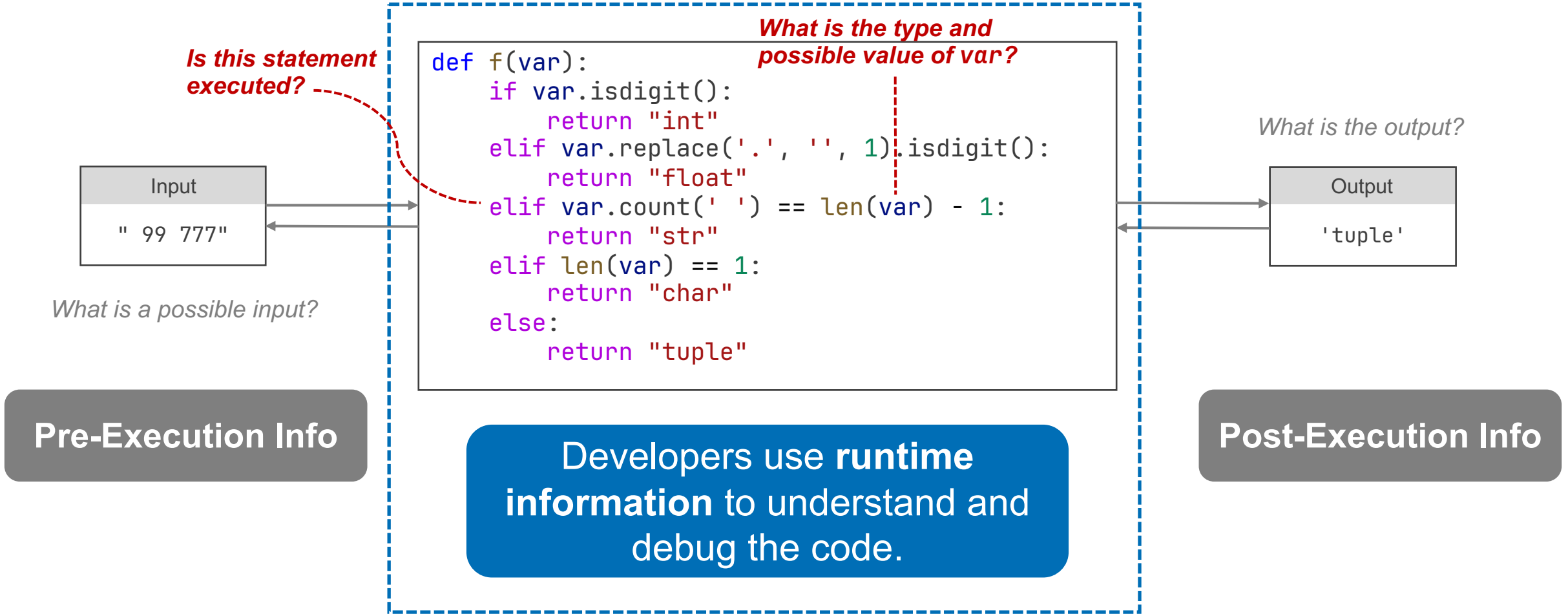
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



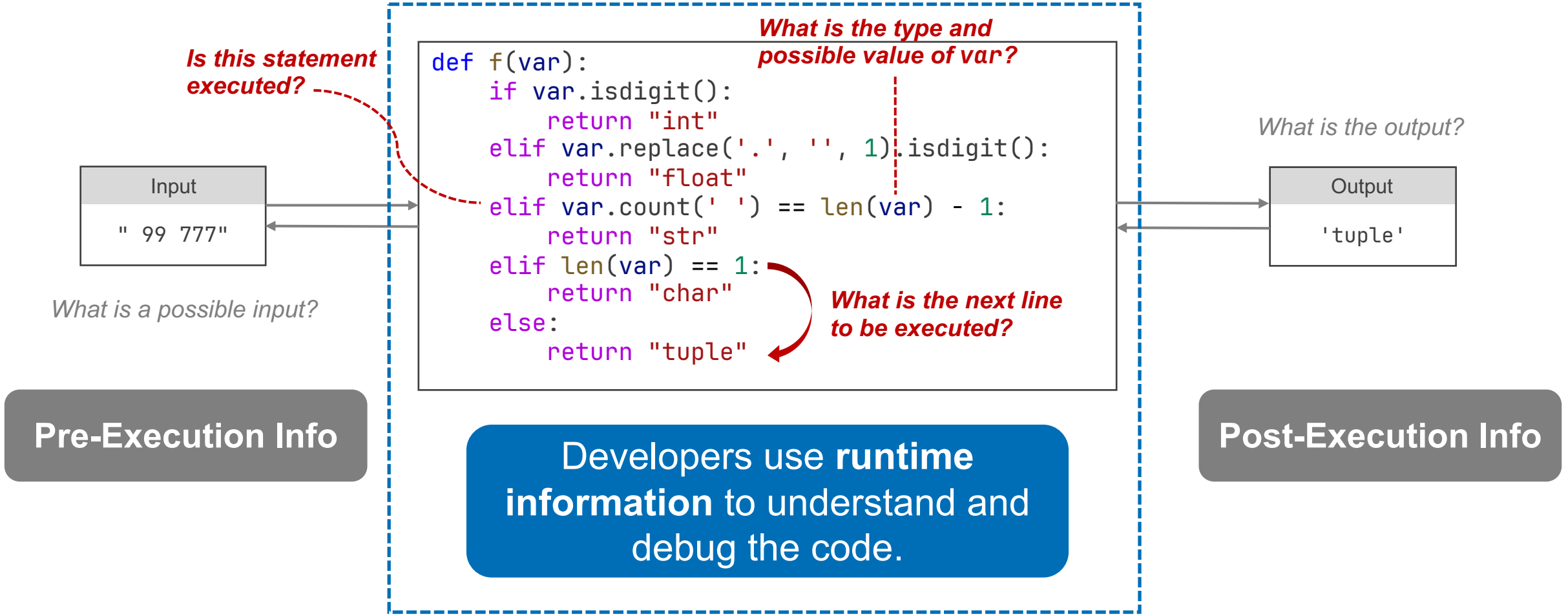
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



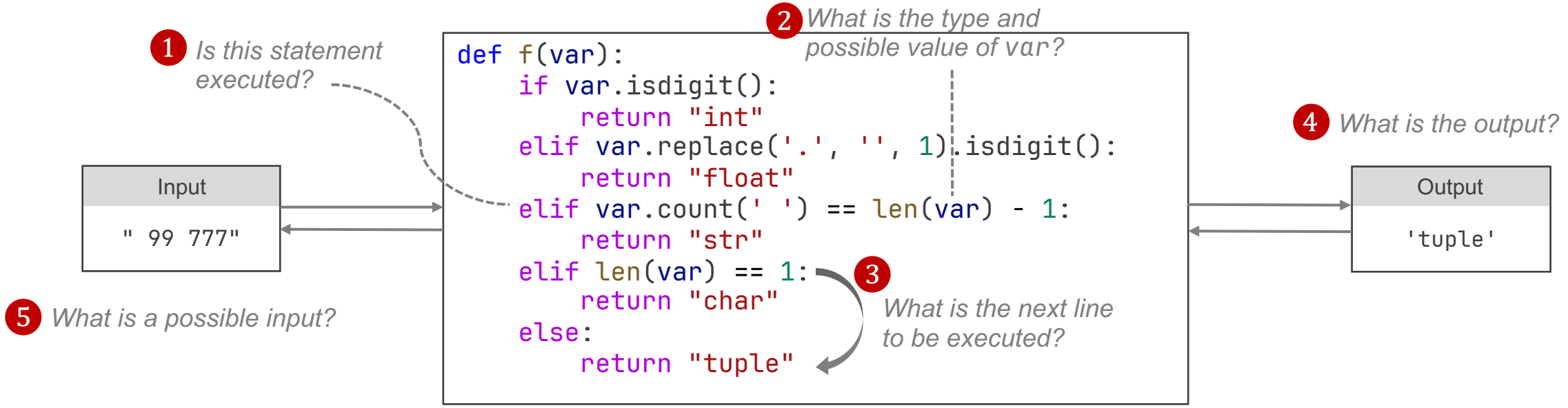
Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*



Previous[1]: 4 5

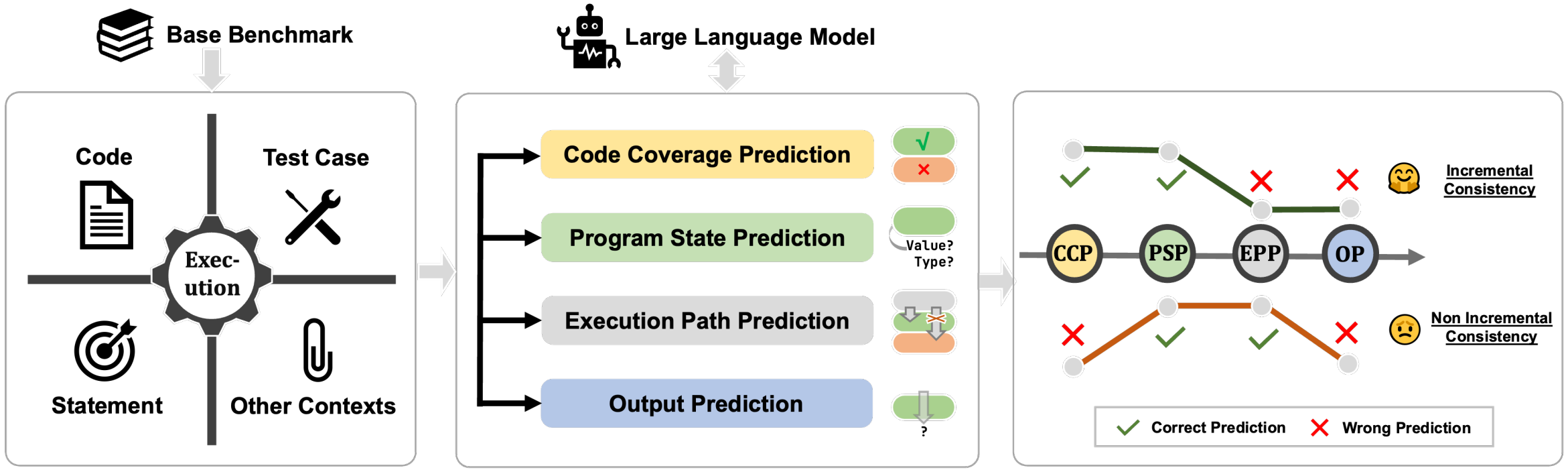
Pre-/Post- Execution Information **Only**

REval (Ours): 1 ▸ 2 ▸ 3 ▸ 4

With **Runtime Information & Consistency**

1. Gu et al. "CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution." *ICML 2024*.

REval Framework



1: Runtime Behavior Reasoning

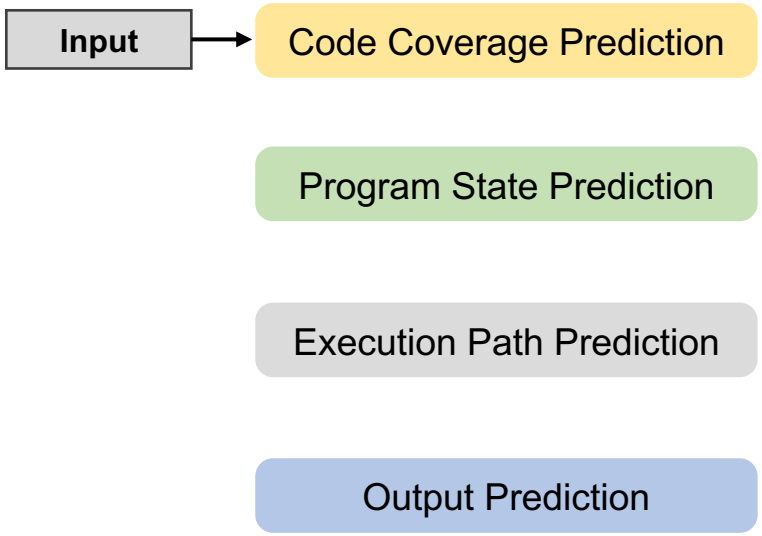
2: *Incremental Consistency* Evaluation

Reason about **four kinds of runtime behavior**.

Evaluate **consistency** with a sequence of previous results.

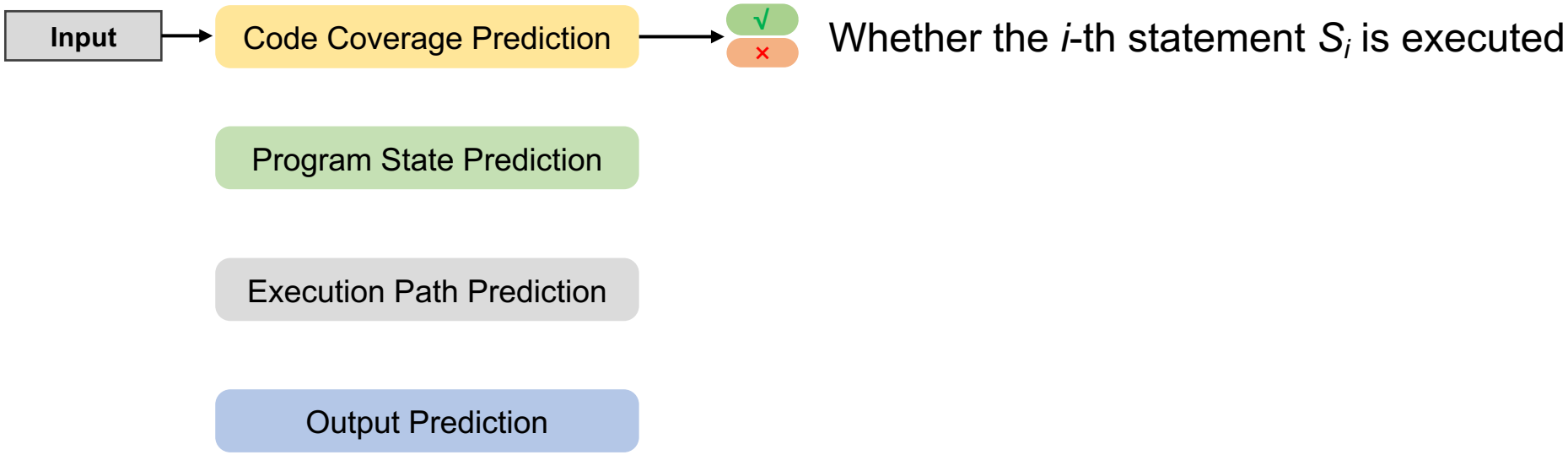
Task Definition & Metrics

Component I: Runtime Behavior Reasoning



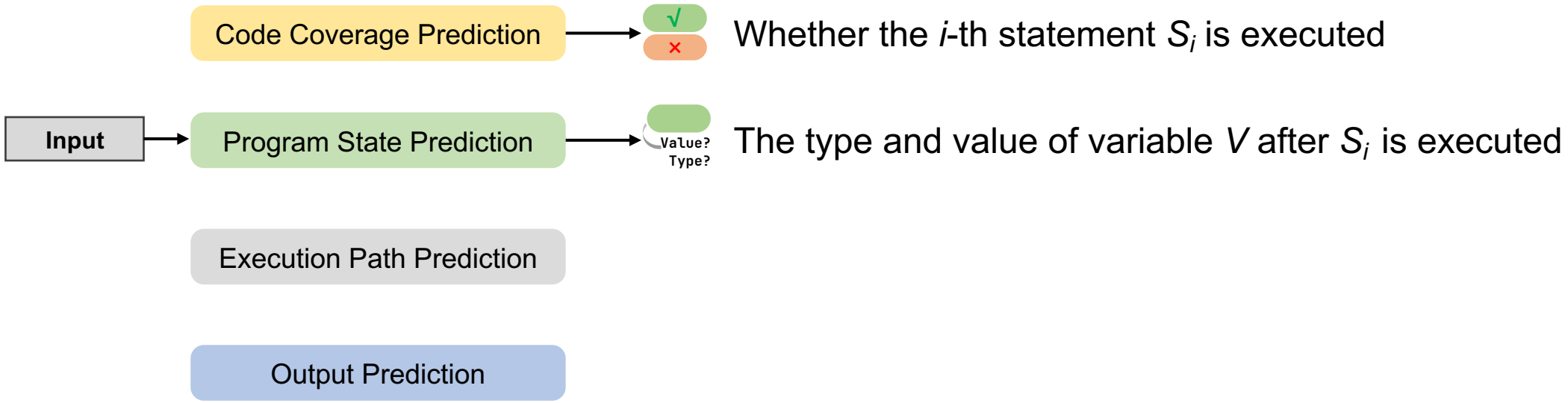
Task Definition & Metrics

Component I: Runtime Behavior Reasoning



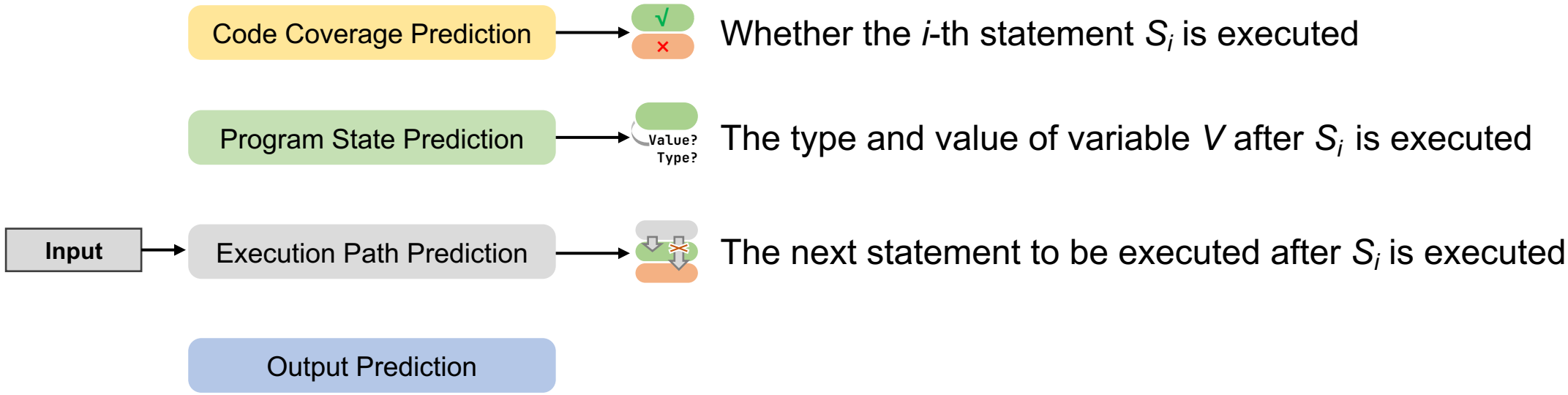
Task Definition & Metrics

Component I: Runtime Behavior Reasoning



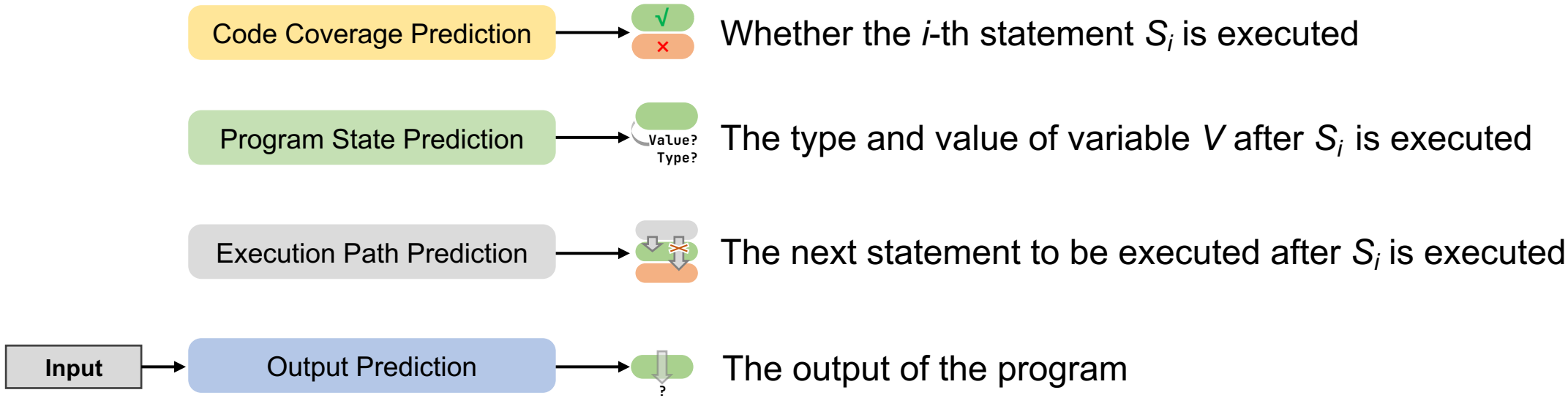
Task Definition & Metrics

Component I: Runtime Behavior Reasoning



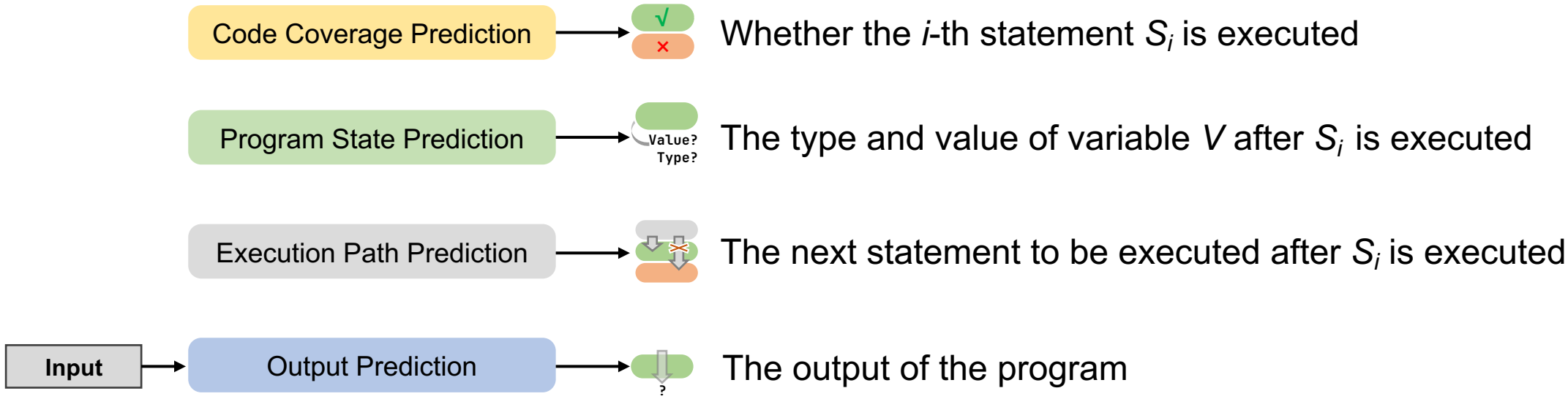
Task Definition & Metrics

Component I: Runtime Behavior Reasoning



Task Definition & Metrics

Component I: Runtime Behavior Reasoning



Whether the *i*-th statement S_i is executed

The type and value of variable *V* after S_i is executed

The next statement to be executed after S_i is executed

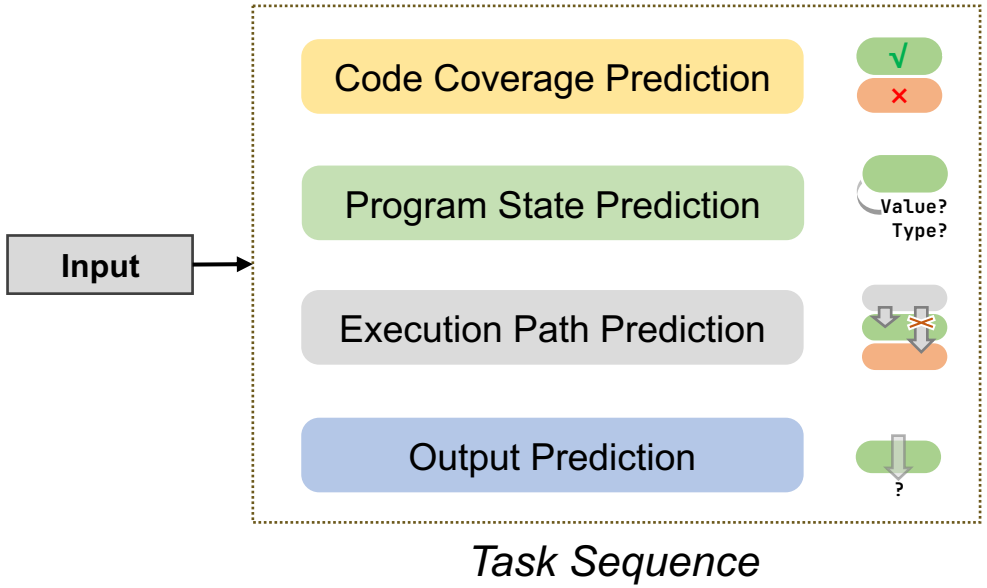
The output of the program

Evaluation Metric(s)

- **Accuracy**: Percentage of correct predictions
- **F1 Score** (Additional for Code Coverage Prediction)

Task Definition & Metrics

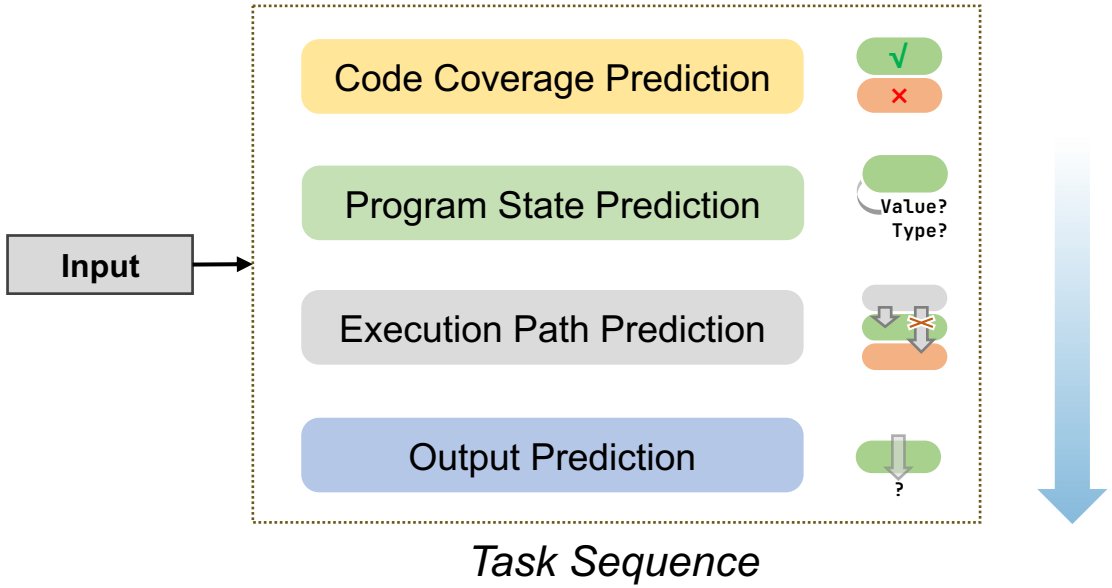
Component II: “Incremental Consistency”



Task Sequence

Task Definition & Metrics

Component II: “Incremental Consistency”

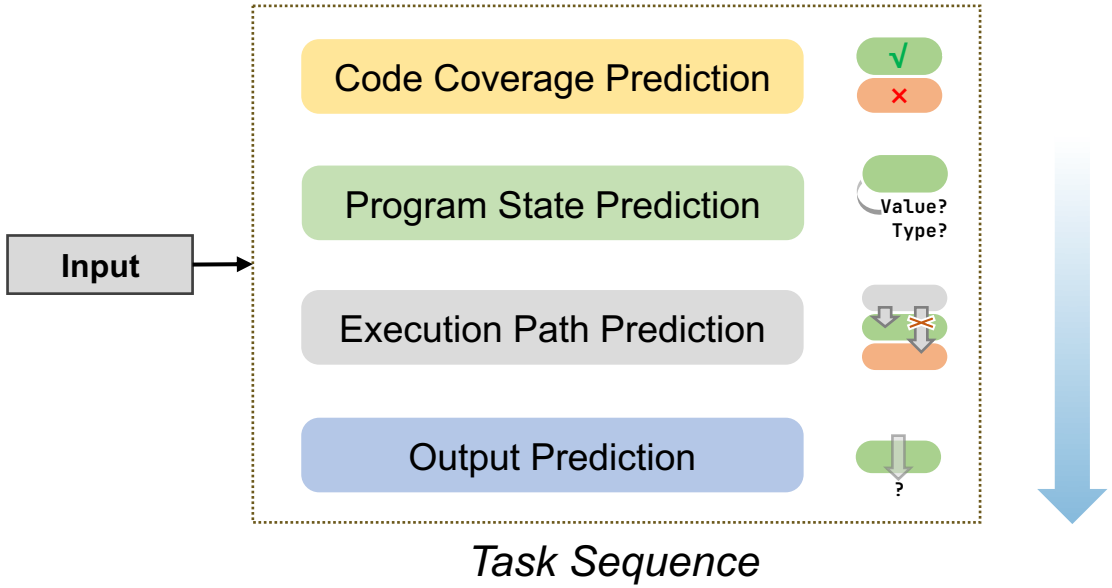


Incremental **knowledge required**

With *incremental* difficulty

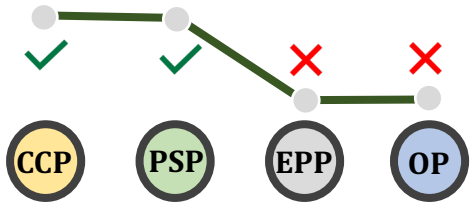
Task Definition & Metrics

Component II: “Incremental Consistency”



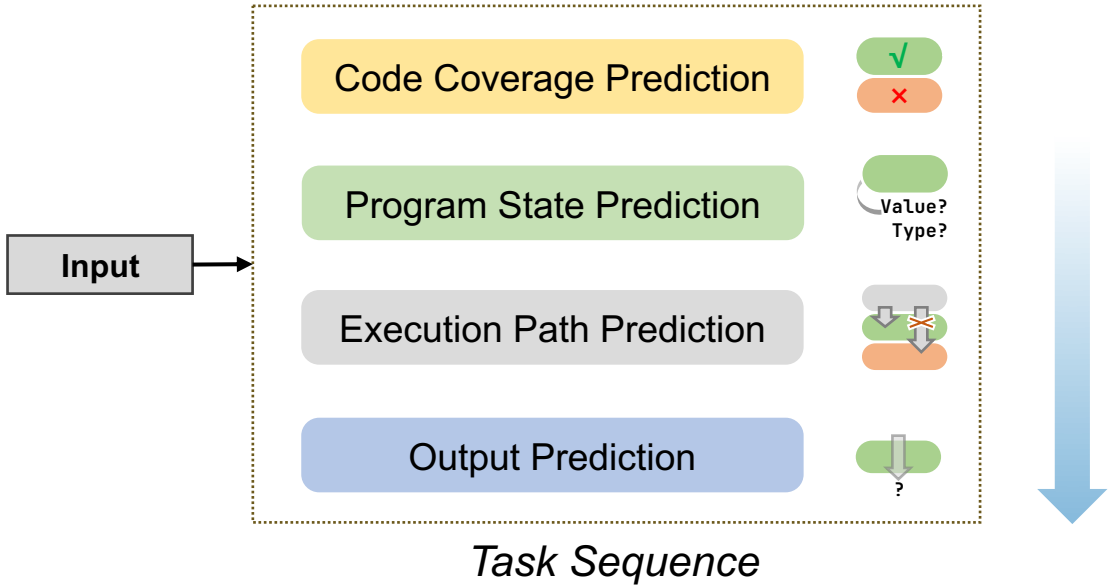
***Incremental* knowledge required**

With *incremental* difficulty



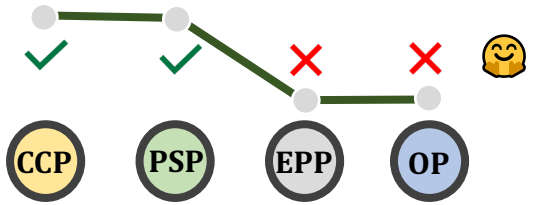
Task Definition & Metrics

Component II: “Incremental Consistency”



Incremental **knowledge required**

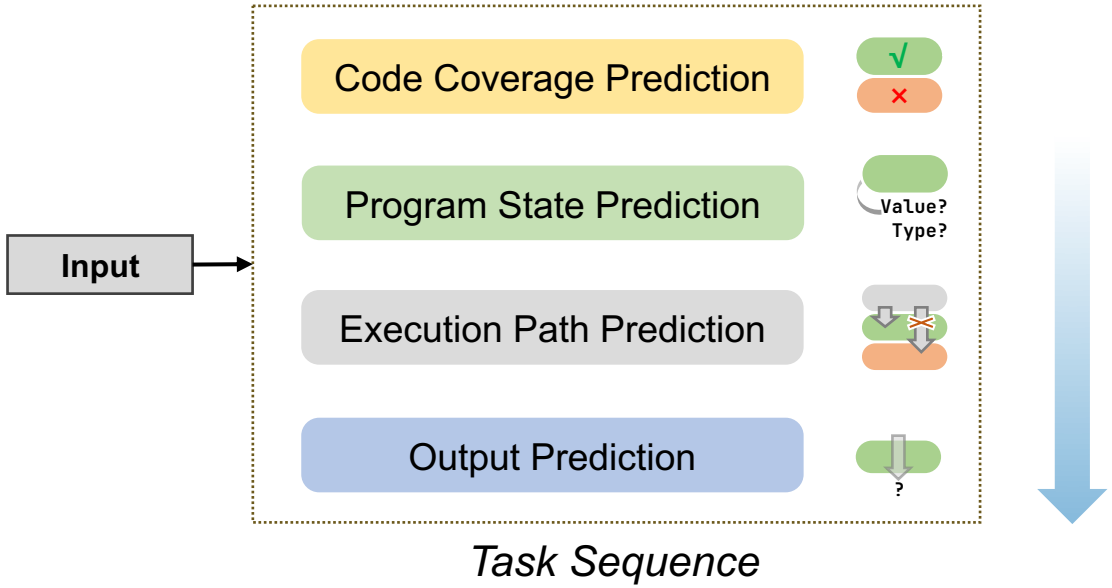
With *incremental* difficulty



$\{1, 1, 0, 0\}$

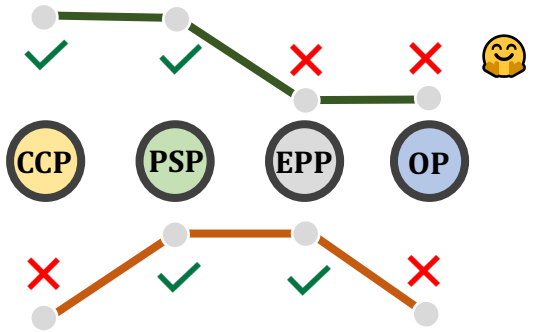
Task Definition & Metrics

Component II: “Incremental Consistency”



Incremental **knowledge required**

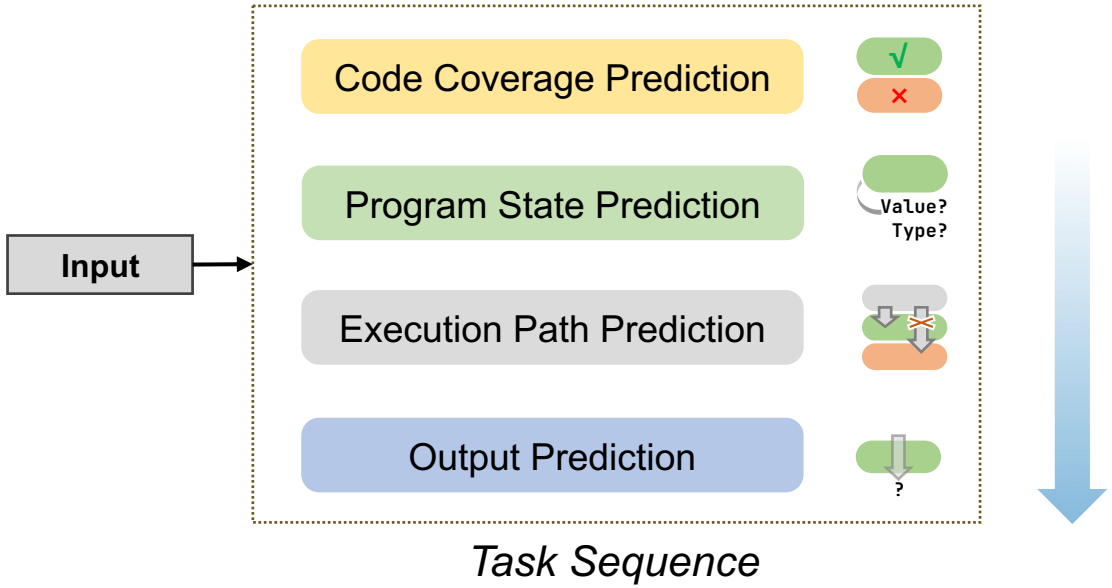
With *incremental* difficulty



$\{1, 1, 0, 0\}$

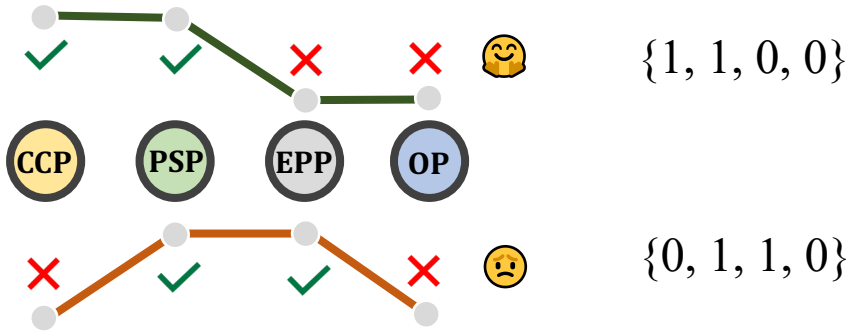
Task Definition & Metrics

Component II: “Incremental Consistency”



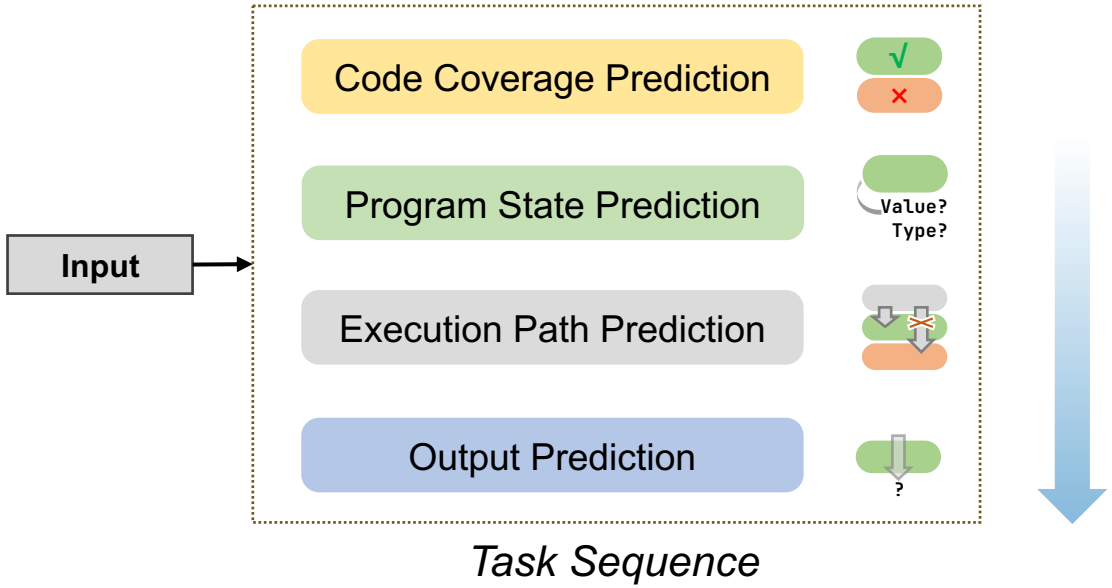
Incremental **knowledge required**

With *incremental* difficulty



Task Definition & Metrics

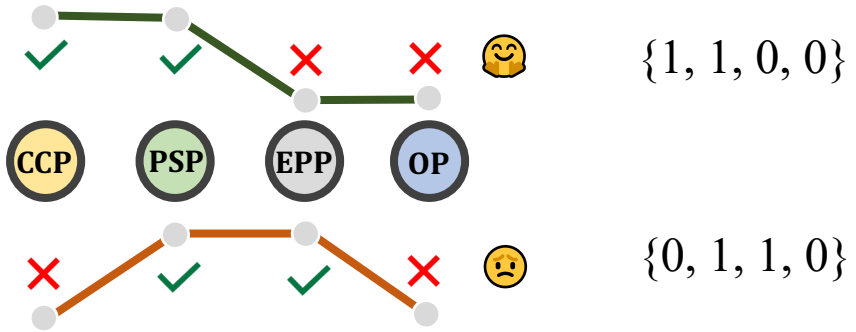
Component II: "Incremental Consistency"



Task Sequence

***Incremental* knowledge required**

With *incremental* difficulty



Evaluation Metric:

$$\text{IC Score} = \frac{100}{N}\sum_{i=1}^{N}\text{IC Score}_i,$$

$$\text{IC Score}_i = \begin{cases} \frac{1}{2^{j-1}}, & \text{if } \mathbf{R}_i = \mathbf{S}_j, j \in \{1,2,3,4\} \\ 0, & \text{otherwise} \end{cases}$$

Result sequences *S* with *"Incremental Consistency"*:

$$\{\{1,1,1,1\},\{1,1,1,0\},\{1,1,0,0\},\{1,0,0,0\}\}$$

Benchmark Construction

A. **Problem Construction: How to build the problems?**

B. **Runtime Behavior Extraction: How to get the ground truth?**

```
def f(var):
    if var.isdigit():
        return "int"
    elif var.replace('.', '', 1).isdigit():
        return "float"
    elif var.count(' ') == len(var) - 1:
        return "str"
    elif len(var) == 1:
        return "char"
    else:
        return "tuple"
```

Benchmark Construction

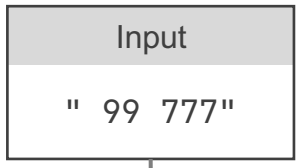
A. **Problem Construction: How to build the problems?**

B. **Runtime Behavior Extraction: How to get the ground truth?**

```python
def f(var):
    if var.isdigit():
        return "int"
    elif var.replace('.', '', 1).isdigit():
        return "float"
    elif var.count(' ') == len(var) - 1:
        return "str"
    elif len(var) == 1:
        return "char"
    else:
        return "tuple"
```

A. Using **canonical solutions** in code generation benchmarks

Benchmark Construction



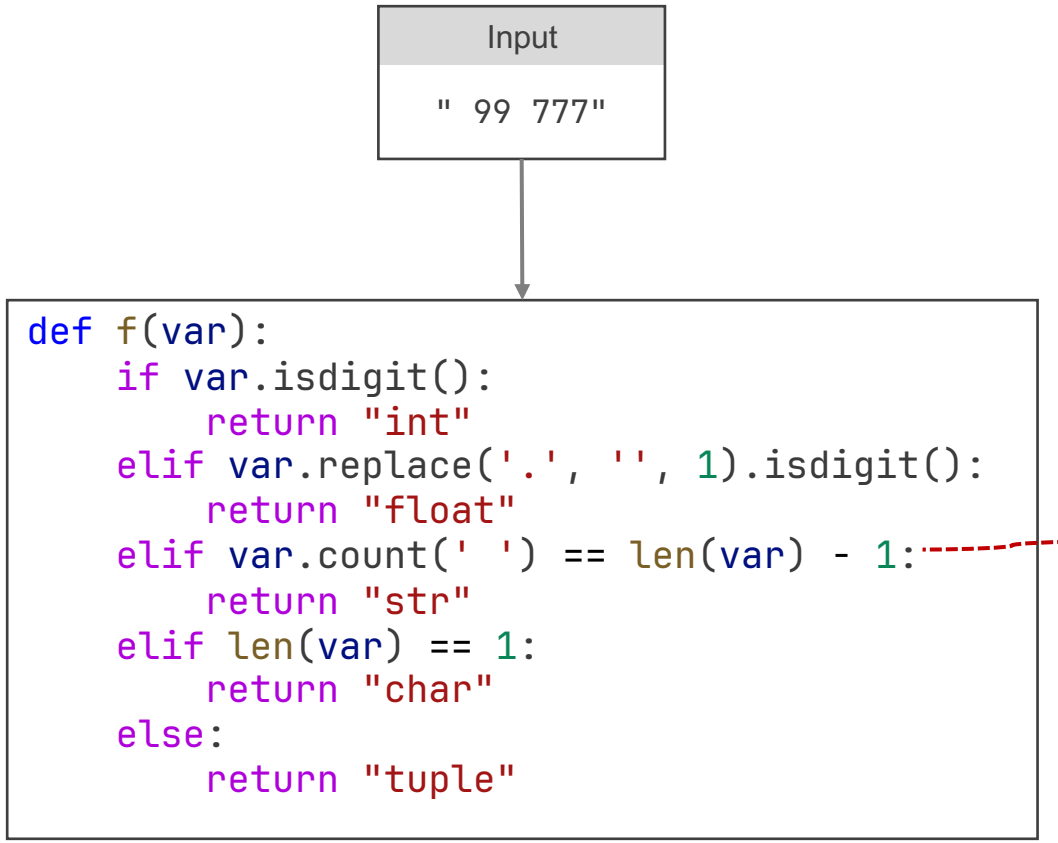
A. From **test cases** in original benchmarks

```
def f(var):
    if var.isdigit():
        return "int"
    elif var.replace('.', '', 1).isdigit():
        return "float"
    elif var.count(' ') == len(var) - 1:
        return "str"
    elif len(var) == 1:
        return "char"
    else:
        return "tuple"
```

A. Using **canonical solutions** in code generation benchmarks

A. **Problem Construction: How to build the problems?**
B. **Runtime Behavior Extraction: How to get the ground truth?**

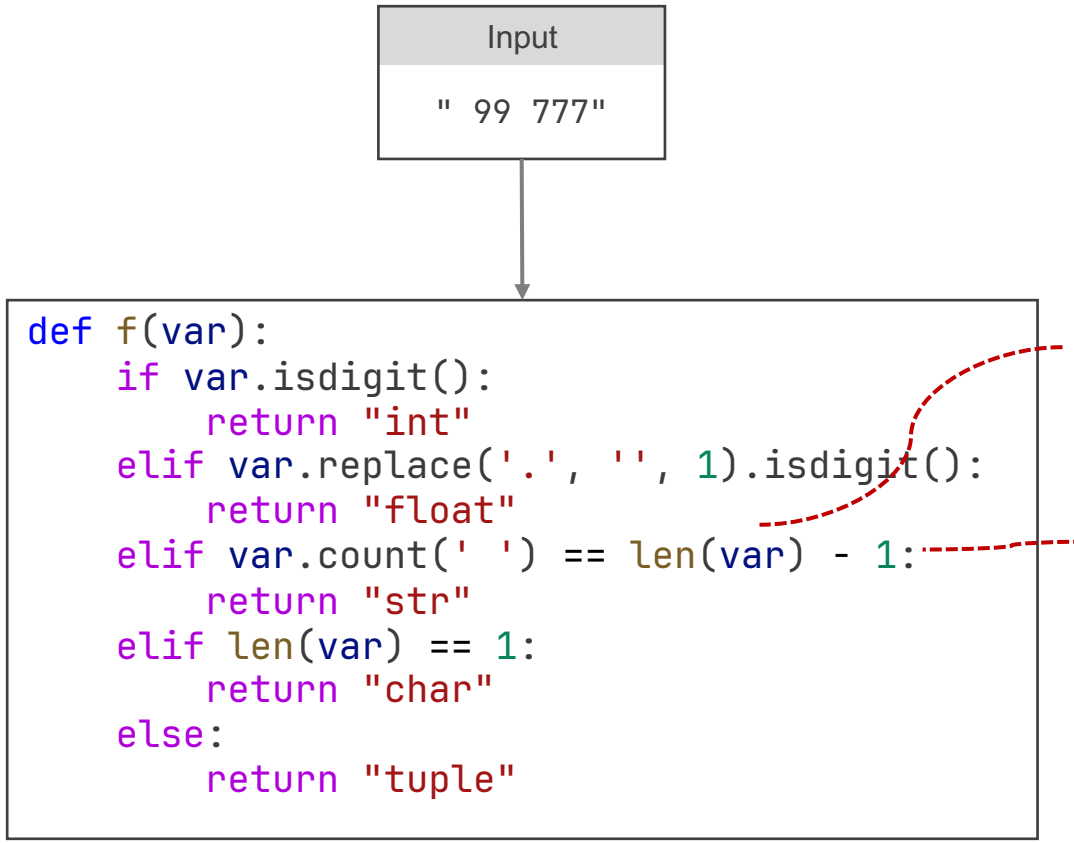
Benchmark Construction



A. **Problem Construction: How to build the problems?**

B. **Runtime Behavior Extraction: How to get the ground truth?**

Benchmark Construction

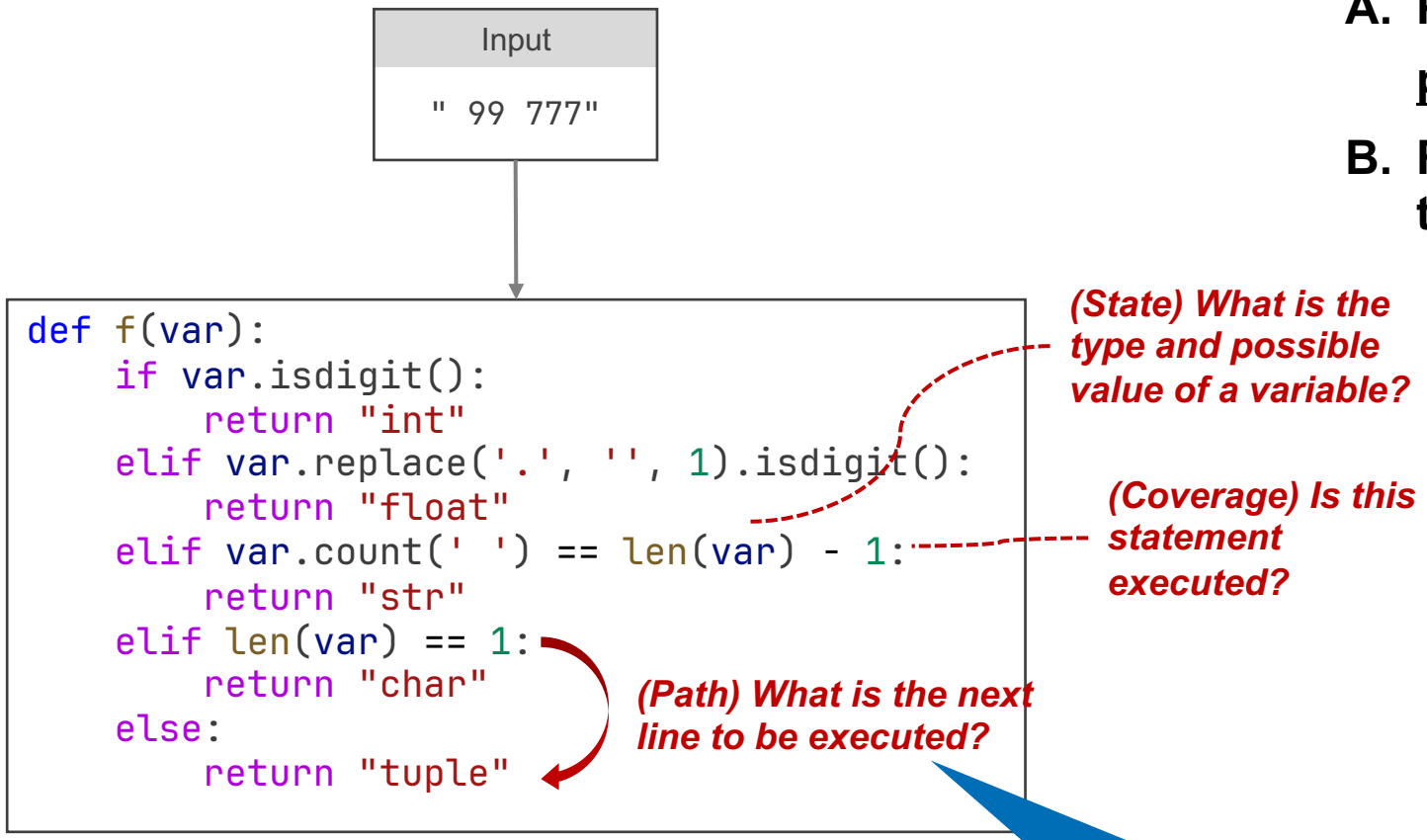


A. **Problem Construction: How to build the problems?**

B. **Runtime Behavior Extraction: How to get the ground truth?**

A. Locate variables in assignment and return, and variables with modified value
B. Inspect runtime frames

Benchmark Construction



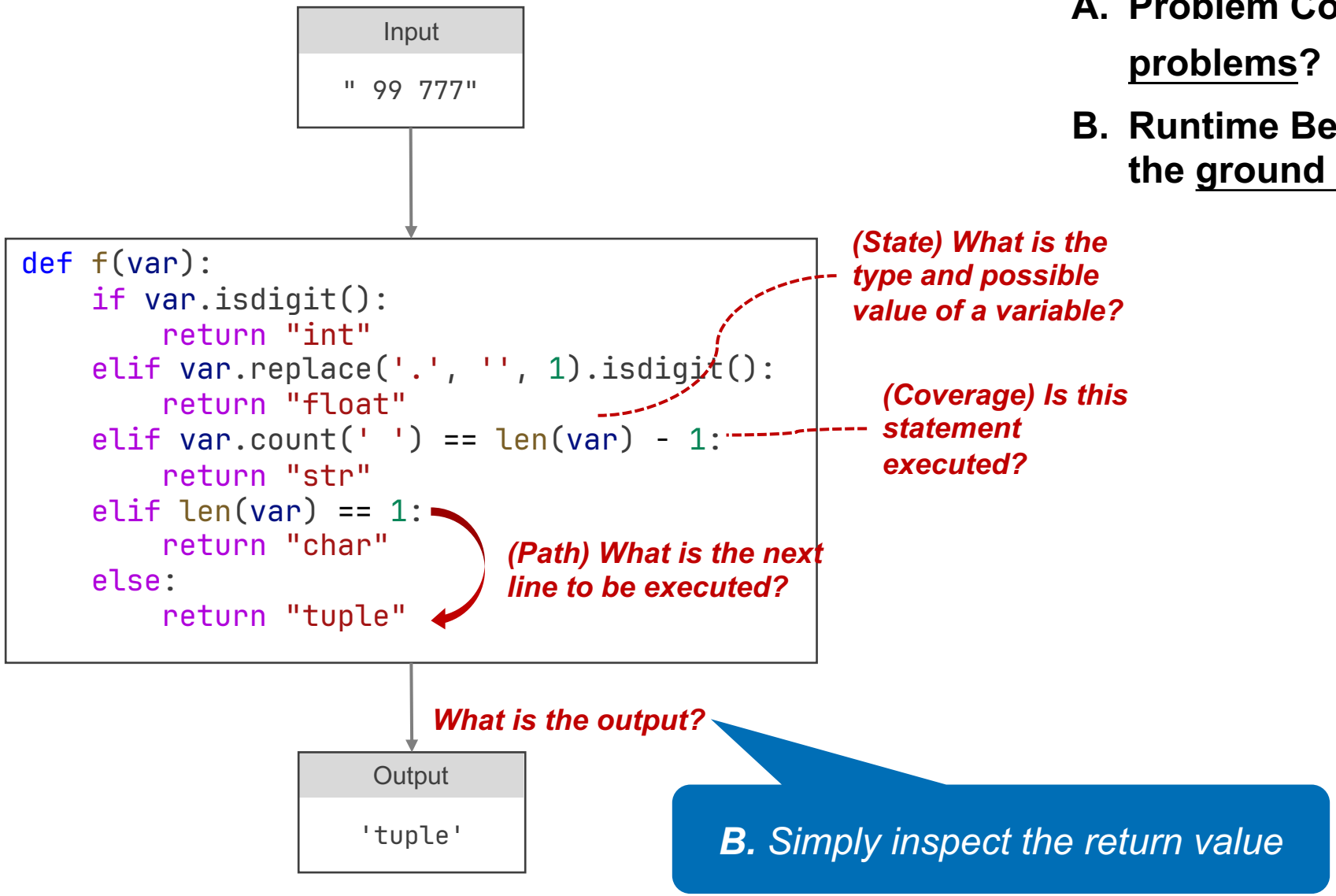
A. **Problem Construction: How to build the problems?**

B. **Runtime Behavior Extraction: How to get the ground truth?**

A. Use the same line numbers in previous tasks

B. Inspect code execution trace

Benchmark Construction



A. **Problem Construction: How to build the problems?**

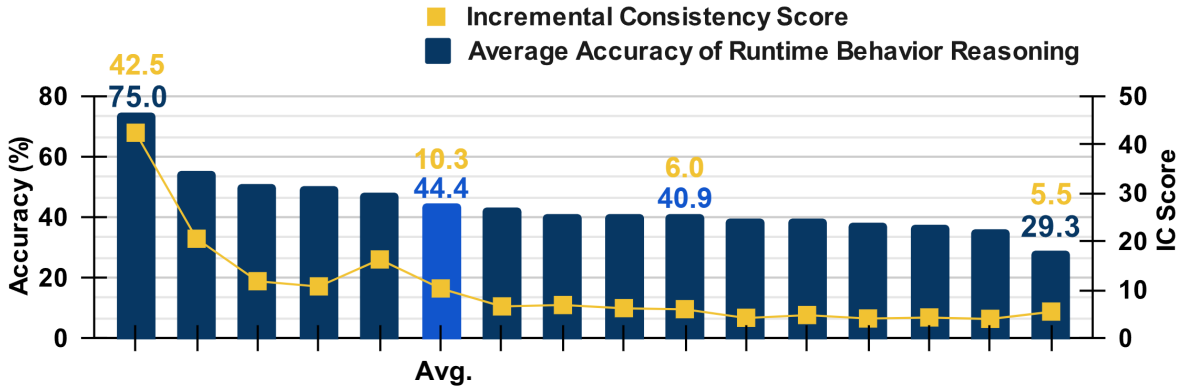
B. **Runtime Behavior Extraction: How to get the ground truth?**

B. *Simply inspect the return value*

Results

*Performance on **Runtime Behavior Reasoning** and **Incremental Consistency Evaluation***

Model	CCP		PSP	EPP	OP	Acc. Avg.	IC
	Acc. (%)	F1	Acc. (%)	Acc. (%)	Acc. (%)	(%)	Score
CodeLlama-7B-Base	54.3±0.5	56.1±0.5	25.0±0.5	5.6±0.3	58.2±1.6	35.8	4.0±0.2
CodeLlama-7B-Python	55.5±0.7	62.7±0.6	31.3±1.0	8.7±0.7	62.3±1.0	39.4	4.8±0.2
CodeLlama-7B-Instruct	55.6±0.9	47.2±1.2	25.1±0.5	10.8±0.2	62.6±0.8	38.5	4.1±0.1
CodeLlama-13B-Instruct	61.0±0.7	66.4±0.6	32.5±0.4	14.4±0.4	64.5±1.1	43.1	6.6±0.3
CodeLlama-34B-Instruct	61.5±0.5	70.1±0.4	47.5±0.6	29.2±0.4	65.9±1.1	51.0	11.8±0.3
StarCoder2-3B	54.8±0.7	58.2±0.5	29.0±0.7	6.5±0.5	58.8±0.8	37.3	4.3±0.3
StarCoder2-7B	55.1±0.7	63.8±0.6	34.2±0.7	5.0±0.4	63.9±0.8	39.6	4.2±0.3
StarCoder2-15B	58.9±0.8	64.6±0.8	43.5±0.3	28.0±0.5	71.5±1.1	50.5	10.7±0.4
Magicoder-CL	58.7±1.4	61.2±1.8	30.1±0.5	15.5±1.1	60.4±1.4	41.2	6.2±0.3
Magicoder-S-CL	60.3±1.1	69.9±0.8	31.4±0.4	9.8±0.4	62.3±1.2	40.9	6.0±0.2
Gemma-2B-It	52.7±0.4	31.0±0.6	13.5±0.5	7.3±0.5	43.9±1.5	29.3	5.5±0.2
Gemma-7B-It	66.3±0.3	75.2±0.1	32.1±0.1	8.4±0.4	57.9±0.7	41.2	6.9±0.2
Mistral-7B-Instruct	69.5±0.2	75.9±0.2	35.2±0.3	35.8±0.4	51.5±0.7	48.0	16.3±0.3
GPT-3.5-Turbo	61.8	64.0	51.6	48.6	60.7	55.7	20.6
GPT-4-Turbo	**88.4**	**89.8**	**71.4**	**57.7**	**82.6**	**75.0**	**42.5**
Average	61.0	63.7	35.6	19.4	61.8	44.4	10.3
CodeLlama-7B-Instruct (CoT)	57.5	59.2	33.4	21.4	55.8	42.2	7.5



GPT-4-Turbo shows superior performance on both components[1].

Execution Path Prediction is the hardest among the four tasks.

CoT helps code reasoning.

Code LLMs do not exhibit obvious leading advantage over general LLMs of the same size.

Models with worse runtime reasoning abilities could show better Incremental Consistency.

1. As of March 2024

Case Study

```
def largest_prime_factor(n: int):
    """Return the largest prime factor of n.
Assume n > 1 and is not a prime.
    >>> largest_prime_factor(13195)
    29
    >>> largest_prime_factor(2048)
    2
    """
    def is_prime(k):
        if k < 2:
            return False
        for i in range(2, k - 1):
            if k % i == 0:
                return False
        return True
    largest = 1
√   for j in range(2, n + 1):
        if n % j == 0 and is_prime(j):
            largest = max(largest, j)
×   return largest
```

GPT-4-Turbo

GPT-3.5-Turbo

CodeLlama-34B-Instruct

Input: 15

Question:
What is the next line to be executed after line "largest = max(largest, j)"?

Manual Analysis

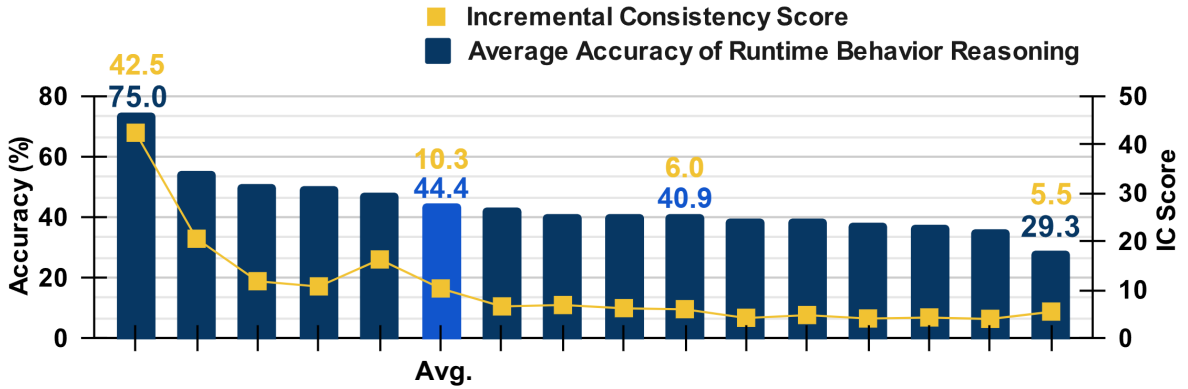
- Range of `j`: $[2, 15]$
- Possible value of j if `largest = max(largest, j)` can be executed: $\{3,\ 5\}$
- Loop is not terminated, if `largest = max(largest, j)` is be executed

Next Line is
`for j in range(2, n + 1):`

Results

*Performance on **Runtime Behavior Reasoning** and **Incremental Consistency Evaluation***

Model	CCP		PSP	EPP	OP	Acc. Avg.	IC
	Acc. (%)	F1	Acc. (%)	Acc. (%)	Acc. (%)	(%)	Score
CodeLlama-7B-Base	54.3±0.5	56.1±0.5	25.0±0.5	5.6±0.3	58.2±1.6	35.8	4.0±0.2
CodeLlama-7B-Python	55.5±0.7	62.7±0.6	31.3±1.0	8.7±0.7	62.3±1.0	39.4	4.8±0.2
CodeLlama-7B-Instruct	55.6±0.9	47.2±1.2	25.1±0.5	10.8±0.2	62.6±0.8	38.5	4.1±0.1
CodeLlama-13B-Instruct	61.0±0.7	66.4±0.6	32.5±0.4	14.4±0.4	64.5±1.1	43.1	6.6±0.3
CodeLlama-34B-Instruct	61.5±0.5	70.1±0.4	47.5±0.6	29.2±0.4	65.9±1.1	51.0	11.8±0.3
StarCoder2-3B	54.8±0.7	58.2±0.5	29.0±0.7	6.5±0.5	58.8±0.8	37.3	4.3±0.3
StarCoder2-7B	55.1±0.7	63.8±0.6	34.2±0.7	5.0±0.4	63.9±0.8	39.6	4.2±0.3
StarCoder2-15B	58.9±0.8	64.6±0.8	43.5±0.3	28.0±0.5	71.5±1.1	50.5	10.7±0.4
Magicoder-CL	58.7±1.4	61.2±1.8	30.1±0.5	15.5±1.1	60.4±1.4	41.2	6.2±0.3
Magicoder-S-CL	60.3±1.1	69.9±0.8	31.4±0.4	9.8±0.4	62.3±1.2	40.9	6.0±0.2
Gemma-2B-It	52.7±0.4	31.0±0.6	13.5±0.5	7.3±0.5	43.9±1.5	29.3	5.5±0.2
Gemma-7B-It	66.3±0.3	75.2±0.1	32.1±0.1	8.4±0.4	57.9±0.7	41.2	6.9±0.2
Mistral-7B-Instruct	69.5±0.2	75.9±0.2	35.2±0.3	35.8±0.4	51.5±0.7	48.0	16.3±0.3
GPT-3.5-Turbo	61.8	64.0	51.6	48.6	60.7	55.7	20.6
GPT-4-Turbo	**88.4**	**89.8**	**71.4**	**57.7**	**82.6**	**75.0**	**42.5**
Average	61.0	63.7	35.6	19.4	61.8	44.4	10.3
CodeLlama-7B-Instruct (CoT)	57.5	59.2	33.4	21.4	55.8	42.2	7.5



GPT-4-Turbo shows superior performance on both components[1].

Execution Path Prediction is the hardest among the four tasks.

CoT helps code reasoning.

Code LLMs do not exhibit obvious leading advantage over general LLMs of the same size.

Models with worse runtime reasoning abilities could show better Incremental Consistency.

1. As of March 2024

Benchmarking Code Reasoning

*Task: Predicting code execution behaviors **without** executing the code.*

REval Framework

Base Benchmark

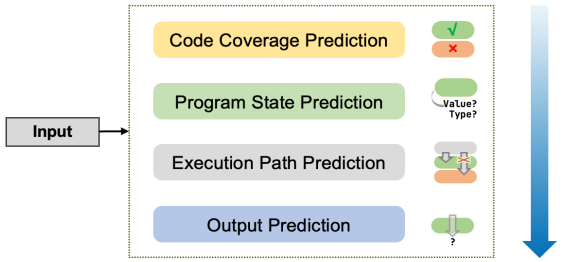
Large Language Model

Paper, Code & Leaderboard

https://r-eval.github.io

Task Definition & Metrics

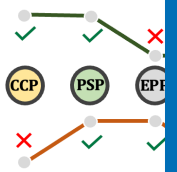
Component II: "Incremental Consistency"



Task Sequence

***Incremental* knowledge**

With *incremental* difficu



Evaluation Metric:

$$\text{IC Score} = \frac{100}{N}\sum_{i=1}^{N}\text{IC Score}_i,$$

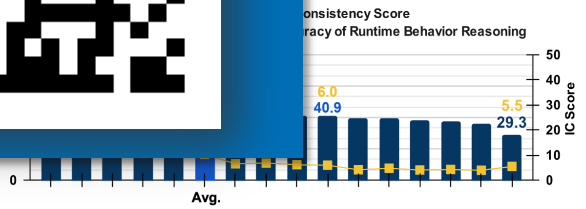
$$\text{IC Score}_i = \begin{cases} \frac{1}{2^{j-1}}, & \text{if } \mathbf{R}_i = \mathbf{S}_j, j \in \{1,2,3,4\} \\ 0, & \text{otherwise} \end{cases}$$

Result seque

$\{\{1,1,1,1\},\{$

r Reasoning* and *Incremental Consistency Evaluation

	EPP	OP	Acc. Avg.	IC
	Acc. (%)	Acc. (%)	(%)	Score
	5.6±0.3	58.2±1.6	35.8	4.0±0.2
	8.7±0.7	62.3±1.0	39.4	4.8±0.2
	10.8±0.2	62.6±0.8	38.5	4.1±0.1
	14.4±0.4	64.5±1.1	43.1	6.6±0.3
	29.2±0.4	65.9±1.1	51.0	11.8±0.3
	6.5±0.5	58.8±0.8	37.3	4.3±0.3
	5.0±0.4	63.9±0.8	39.6	4.2±0.3
	28.0±0.5	71.5±1.1	50.5	10.7±0.4
	15.5±1.1	60.4±1.4	41.2	6.2±0.3
	9.8±0.4	62.3±1.2	40.9	6.0±0.2
	7.3±0.5	43.9±1.5	29.3	5.5±0.2
	8.4±0.4	57.9±0.7	41.2	6.9±0.2
	35.8±0.4	51.5±0.7	48.0	16.3±0.3
	48.6	60.7	55.7	20.6
	57.7	**82.6**	**75.0**	**42.5**
	19.4	61.8	44.4	10.3
	21.4	55.8	42.2	7.5



- GPT-4-Turbo shows superior performance on both components[1].
- Execution Path Prediction is the hardest among the four tasks.
- CoT helps code reasoning.
- Code LLMs do not exhibit obvious leading advantage over general LLMs of the same size.
- Models with worse runtime reasoning abilities could show better Incremental Consistency.

1. As of March 2024