# Ensuring correctness of code generation through intermediate representation reasoning

Ruijun Feng,

Program Analysis Group

University of New South Wales (UNSW)

10/05/2024

# Challenges

- **Precision in code**:
- Unlike natural language that tolerate ambiguity, code demands high precision and adherence to specifications.

# Challenges

- For example, Python requires strict indentation, which the model may mis produce.

**Correct indentation**

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

**Incorrect indentation**

```
def factorial(n):
if n == 0:
    return 1
    else:  # Indentation error here
        return n * factorial(n - 1)

print(factorial(5))
```

# Challenges

- **Gap between user's informal description and target code**:
- Code language model (CLM) generate code **autoregressively**. Each output token depends on the previous tokens.

# Challenges

- Using user's informal description to directly generate code with correct logic is hard.

Write a Python function that takes a list of integers and returns a new list containing only the **even numbers** from the input list.

→ CLM →

```
def filter_even_numbers(lst):
    even_numbers = []
    for x in lst:
        if x % 2 != 0:
            even_numbers.append(x)
    return even_numbers
```

**Wrong logic**

# Challenges

- **High computational costs and latency**:
- To generate logic correct code from user’s informal description, it usually require **multiple inference iterations for refinements**.
- Operating CLMs in this way involves significant computational costs and latency.

# Challenges

- Operating CLMs in this way involves significant computational costs and latency.

You made a mistake on the condition. Update your code

CLM

```
def filter_even_numbers(lst):
  even_numbers = []
  for x in lst:
    if x % 2 = 0:
      even_numbers.append(x)
  return even_numbers
```

**Wrong operator**

**Refinement**

# Literature and Existing Efforts

- To enhance correctness of the generation process, previous research has employed a **chain-of-thought (CoT)** methodology to **mimic human reasoning** by solving problems in steps.

# Standard Prompting

Example

- **Model Input**

Q: Your warehouse has 5 pallets of widgets. You purchase 2 more shipments of widgets. Each shipment contains 3 pallets. How many pallets of widgets do you have now?
A: The answer is 11.

Q: Your finance department has $23,000 in the budget. If they allocate $20,000 for a marketing campaign and add $6,000 from other savings, how much is left in the budget?

- **Model Output**

A: The answer is $27,000. ❌

Target Question

# Chain-of-thought prompting

Example with reasoning steps

- **Model Input**

  Q: Your warehouse has 5 pallets of widgets. You purchase 2 more shipments of widgets. Each shipment contains 3 pallets. How many pallets of widgets do you have now?
  A: **The warehouse started with 5 pallets of widgets. 2 shipments of 3 pallets each is 6 pallets. 5 pallets + 6 pallets = 11 pallets.** The answer is 11 pallets.
  Q: Your finance department has $23,000 in the budget. If they allocate $20,000 for a marketing campaign and add $6,000 from other savings, how much is left in the budget?

- **Model Output**

  A: **The finance department originally had $23,000. They allocated $20,000 for a marketing campaign, which leaves them with $23,000 — $20,000 = $3,000. They then added $6,000 from other savings, so the new total is $3,000 + $6,000 = $9,000.** The answer is $9,000. ✓

Reference: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, NeurIPS 2022

# Limitations of Existing Works

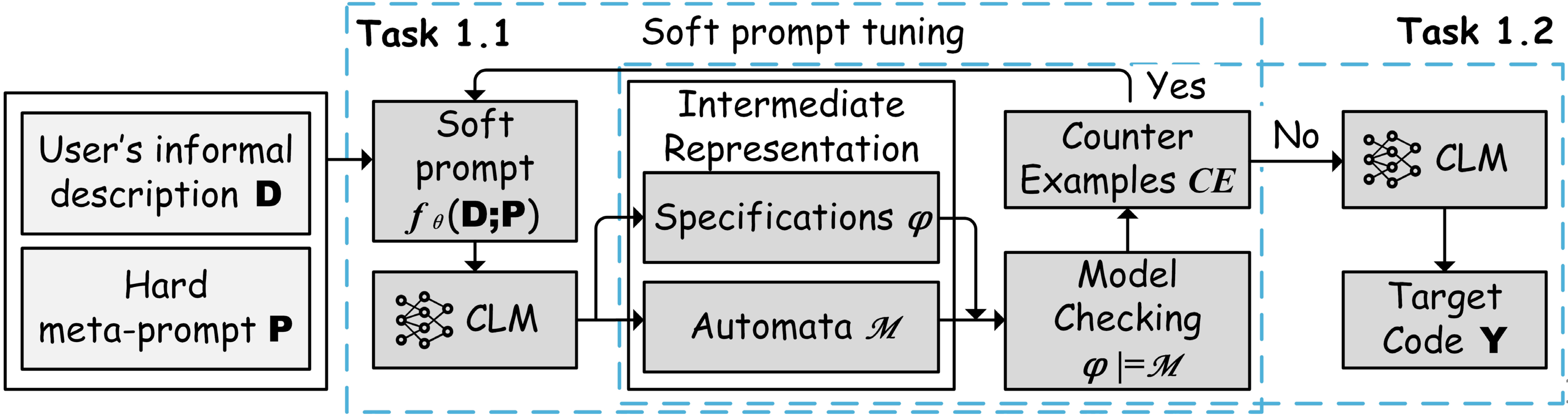
- However, code generation, which often demands algorithmic reasoning. Traditional CoT method faces challenges in **ensuring the correctness of the informal intermediate steps**.

For example:

- **Model Output**
  A: **The finance department originally had $23,000. They allocated $20,000 for a marketing campaign, which leaves them with $23,000 — $20,000 = $~~3~~2,000. They then added $6,000 from other savings, so the new total is $~~3~~2,000 + $6,000 = $~~9~~8,000.** The answer is $**8**,000.

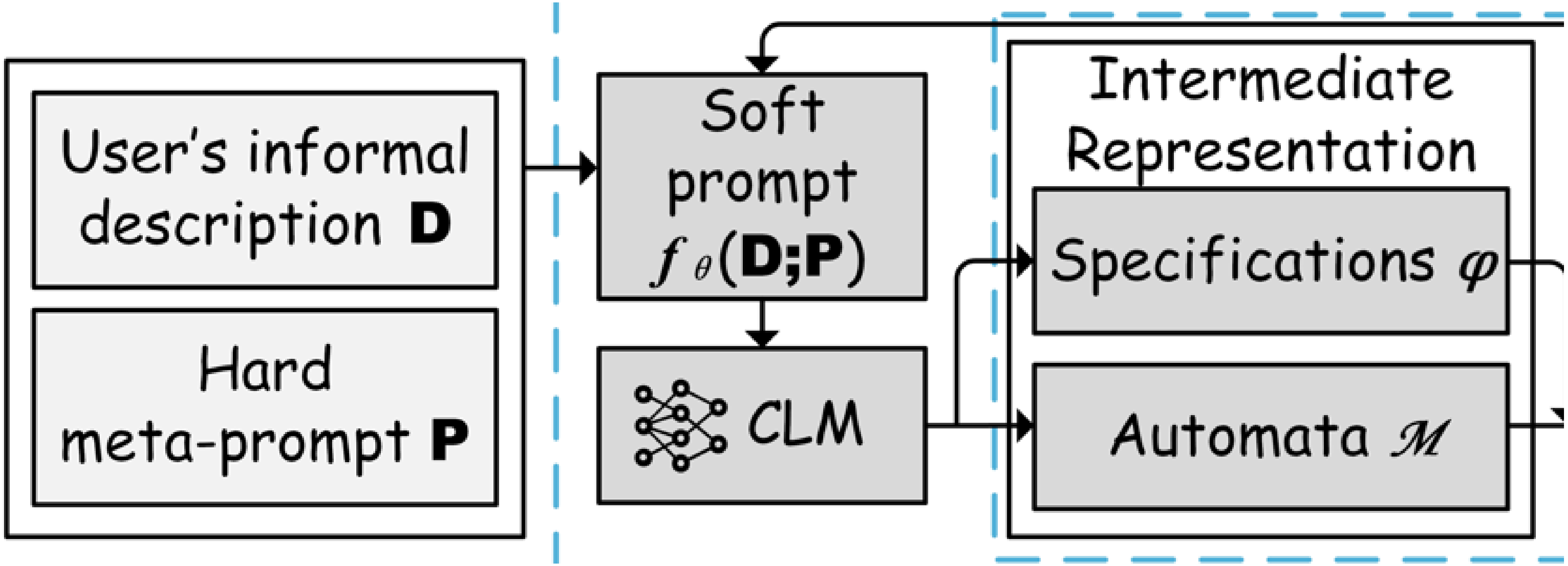
# Our Solution

- A new intermediate representation (IR) reasoning approach for target code generation with the following objectives:
- Objective 1: Ensure the correctness of intermediate steps.
- Objective 2: Generate logic correct code based on the certified intermediate steps.



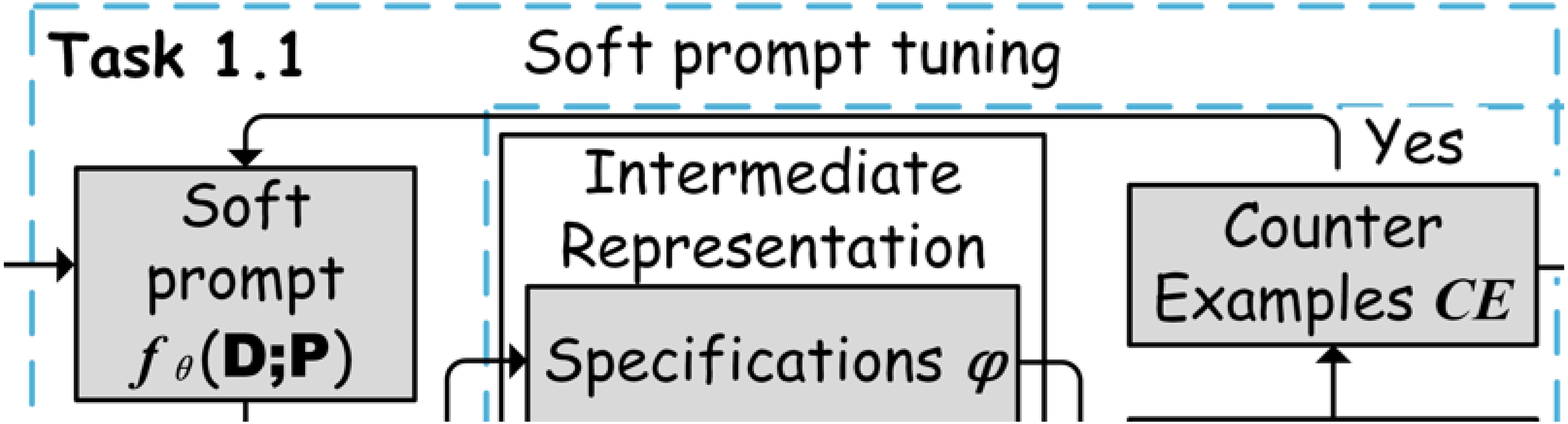
# Our Solution

1. Use a small model $f_\theta$ to translate user's informal descriptions and hard meta prompt (definitions of IR) into soft prompt that guide CLM in generating well-defined specifications and automata.



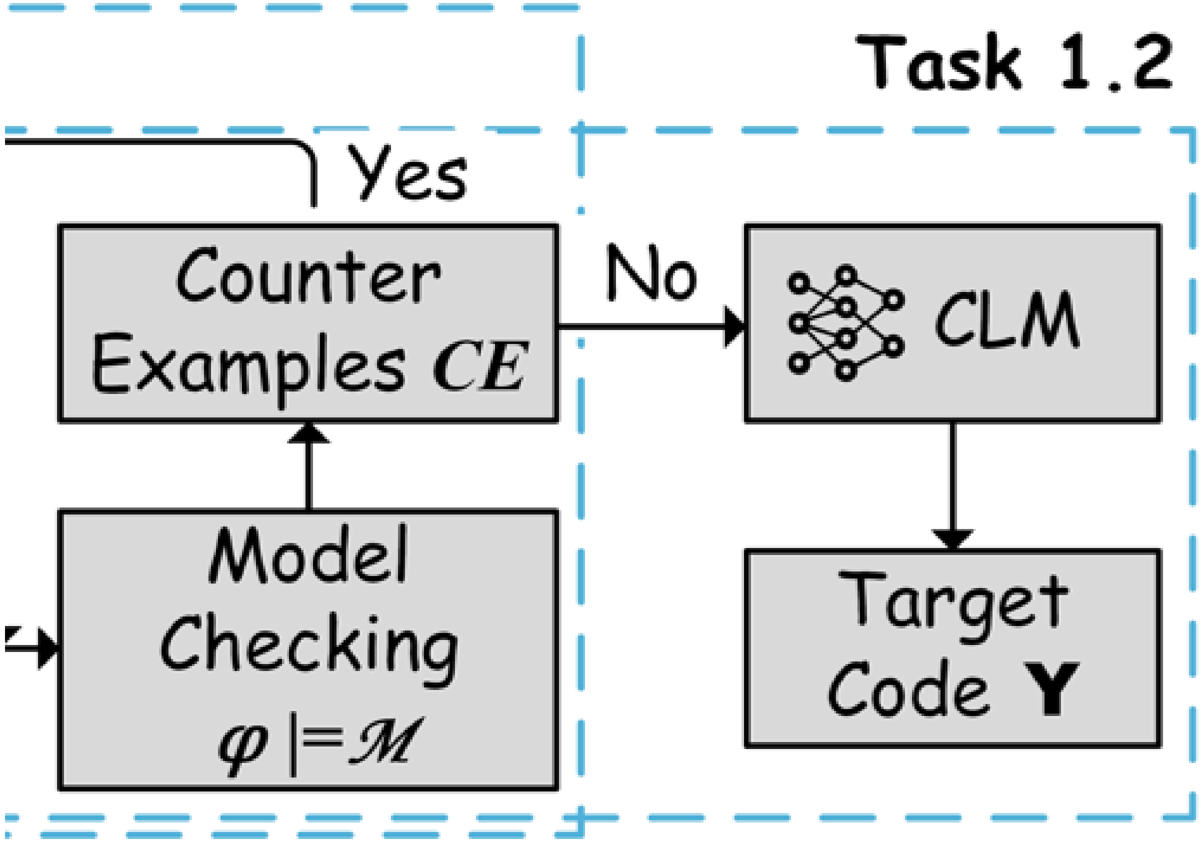
# Our Solution

2. Cross-checking through model checking to ensure state correctness of specifications and automata in a fast, iterative process. If there are counterexamples, then update the soft prompt.



# Our Solution

3. If there are no counterexamples after model checking, then let CLM generate target code based on the certified specifications and automata .



# Evaluation

- Metric
  - Pass Rate: the percentage of test cases that the generated code successfully passes.
  - Code BLEU: compared the textual similarity of the generated code with ground truth.
- Dataset
  - HumanEval—assessing the capabilities of CLM in generating functional and correct code snippets.

# Mainstream Code Language Models

- Starcoder2 (Developed by the BigCode Project 2024):
  - Parameter Sizes: 3B, 7B, and 15B.
  - Open Source: https://huggingface.co/blog/starcoder2.
- CodeGemma (Developed by Google 2024):
  - Parameter Sizes: 2B, 7B.
  - Open Source: https://huggingface.co/collections/google/codegemma-release-66152ac7b683e2667abdee11.
- Code Llama (Developed by Meta 2023):
  - Parameter sizes: 7B, 13B, 34B, 70B.
  - Open Source: https://huggingface.co/codellama.
- DeepSeekCoder (Developed by an Open Source Community 2023)
  - Parameter sizes: 1.3B, 6.7B, 7b, 33B.
  - Open Source: https://huggingface.co/collections/deepseek-ai/deepseek-coder-65f295d7d8a0a29fe39b4ec4.
- PolyCoder (Based on GPT2 2022):
  - Parameter sizes: 2.7B.
  - Open source: https://huggingface.co/NinedayWang/PolyCoder-2.7B.
- Codex (Developed by OpenAI 2023):
  - Parameter sizes: Built on OpenAI's GPT-3, which itself has a wide range of model sizes up to 175B parameters.
  - Close source.