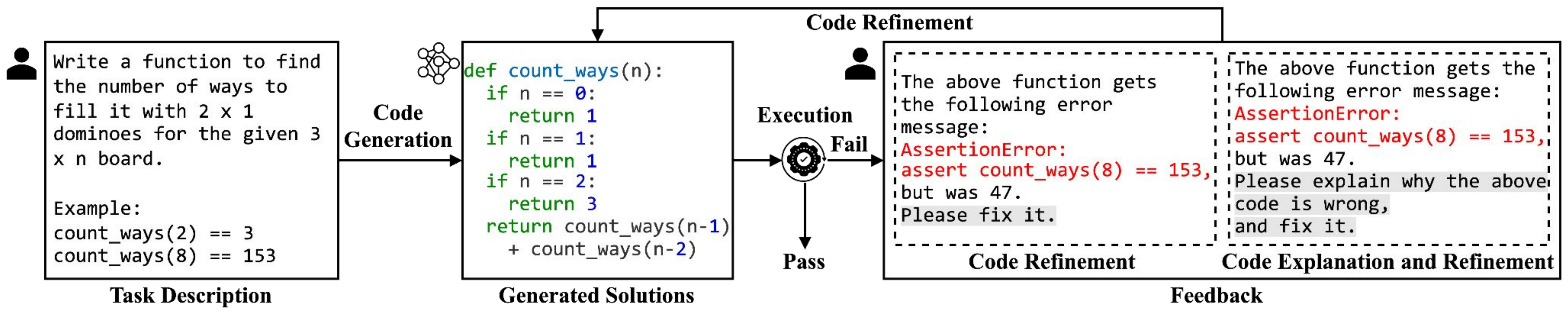
LeDex: Train LLMs to Better Self-Debug and Explain Code

Nan Jiang*, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya Binta Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, Anoop Deoras

1. Motivation: Open-Source LLMs Cannot Self-Debug Effectively

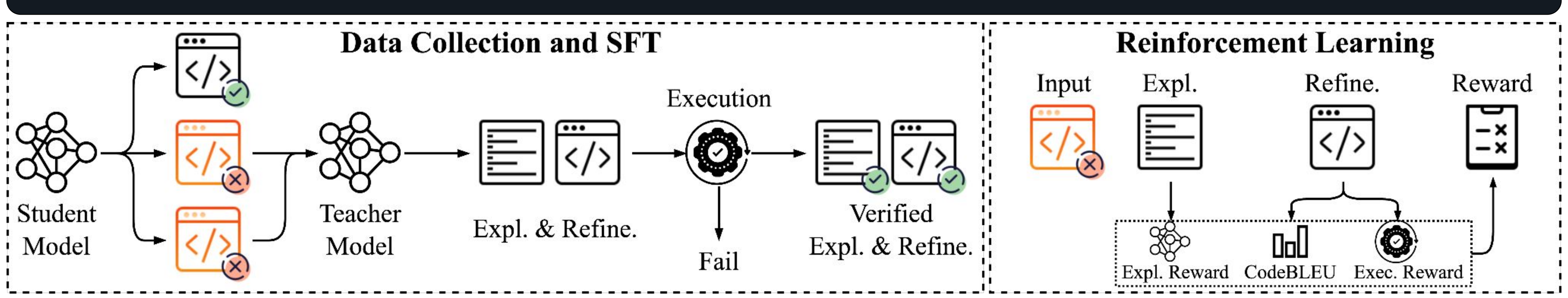


- Self-debug ability is crucial for LLMs to refine their generated code based on execution feedback, which is important to solve complex problems.
- A chain of code explanation and code refinement is more effective for self-debug.
- However, open-source LLMs such as StarCoder and CodeLlama are very limited in self-debug. Their self-debug success rate is as low as 4.43 - 10.20%, while GPT-3.5-Turbo is 28.90%.

This work propose **LeDex**, a training pipeline to improve LLMs self-debug ability. LeDex consists of data collection, supervised fine-tuning and reinforcement learning.

- With GPT-3-5-Turbo as the teacher model, LeDex brings the student model up to **17.79% improvement on pass@1** and **11.18% higher success rate of self-debug**.
- With self-bootstrap data collection where the teacher model is the student model itself, LeDex still brings up to **8.52% improvement on pass@1**.

2. Approach: Data Collection, SFT and RL



Data Collection

- LeDex first samples solutions for tasks in **MBPP** training set, **APPS** and **CodeContests**. These samples solutions are executed to collect the incorrect ones.
- A teacher model (a larger LLM such as GPT-3-5-Turbo, or the student model itself) is prompted to collect code explanation and refinements.
- Only explanation along with refinements that pass the test cases are retained as SFT training data.

Supervised Fine-Tuning

- We experiment with **StarCoder-15B**, **CodeLlama-7B** and **CodeLlama-13B** as the student models, using the collected wrong code, explanation and refined code triples to train the LLMs.

Reinforcement Learning

- RL considers explanation and refinement reward. **Explanation reward** encourages code explanations that are semantically similar to the "best explanation" which leads to correct refinement. **Refinement reward** encourages code refinements that pass more test cases and are similar to the ground-truth code.

CodeBLEU similarity with correct code — Pass rate of unit tests — Semantical similarity with "best explanation"

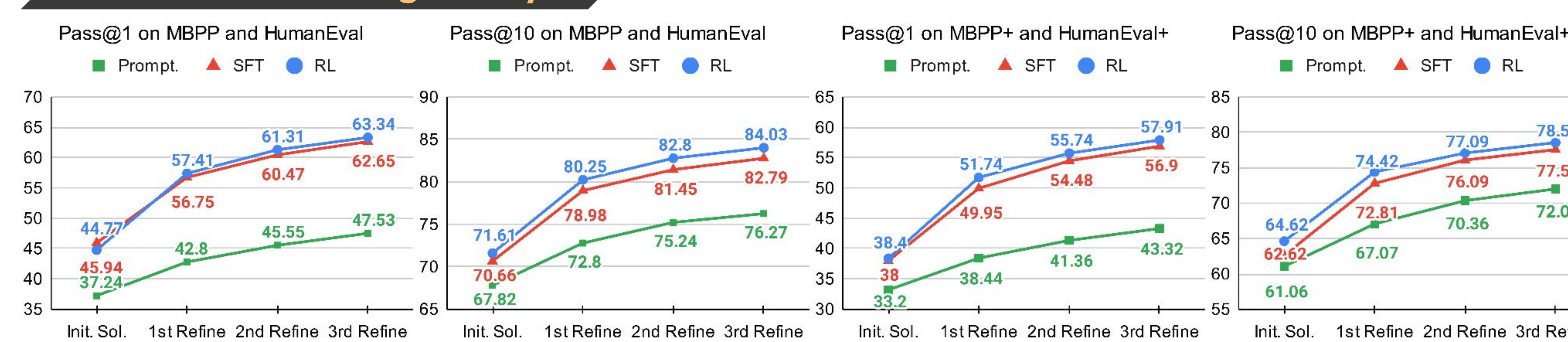
$$\underbrace{R(r)}_{\text{Refinement Reward}} = 5 \cdot (S_{cb}(r) + S_{ut}(r)) - 5 \qquad \underbrace{R(e)}_{\text{Explanation Reward}} = \frac{1}{3}(50 \cdot S_{ex}(e) - 35)$$

3. Evaluation

GPT-3.5-Turbo as Teacher

Model	Approaches		MBPP		HumanEval		MBPP plus		HumanEval plus	
			pass@1	pass@10	pass@1	pass@10	pass@1	pass@10	pass@1	pass@10
CodeLlama-7B	Prompting	Initial Solution	38.21	67.24	34.27	69.60	37.18	61.23	27.40	60.81
		Explain + Refine	43.42	72.09	40.13	74.95	42.46	67.41	32.49	66.58
	LeDex SFT	Initial Solution	48.87	70.89	36.99	69.95	42.97	62.69	30.76	62.52
		Explain + Refine	**57.98**	77.92	52.98	82.22	51.55	70.94	47.62	75.54
	LeDex RL	Initial Solution	46.54	71.54	39.38	71.84	41.46	63.68	33.95	65.98
		Explain + Refine	57.92	**78.97**	**55.84**	**84.14**	**52.90**	**71.80**	**50.04**	**78.25**

Iterative Self-Debug Ability



CodeLlama-7B Self-Bootstrap

Model	Approaches		MBPP		HumanEval		MBPP plus		HumanEval plus	
			pass@1	pass@10	pass@1	pass@10	pass@1	pass@10	pass@1	pass@10
CodeLlama-7B	LeDex SFT	Initial Solution	45.83	69.24	39.85	68.83	41.78	61.77	33.25	61.50
		Explain + Refine	51.80	**74.99**	45.70	74.72	45.94	65.77	39.10	67.33
	LeDex RL	Initial Solution	46.28	68.87	39.90	69.49	41.61	61.29	33.66	62.17
		Explain + Refine	**52.34**	74.60	**46.90**	**75.70**	**46.10**	**65.99**	**40.79**	**68.50**

- LeDex brings **significant improvement** on LLMs self-debug ability.
- LeDex is **model-agnostic**, using either larger teacher model or self-bootstrap for data collection improves.
- LLMs trained with LeDex show strong and **better iterative self-debug ability**.

4. Case Study: LeDex Enables LLMs to Correctly Explain and Debug Code

User: Create a function that takes 3 numbers. Returns true if one of the numbers is equal to the sum of the other two, and all numbers are integers. Returns false in any other cases.
Examples:
any_int(5, 2, 7) ➞ True
any_int(3.6, -2.2, 2) ➞ False

Assistant:

```
def any_int(x, y, z):
    if x + y == z or x + z == y or y + z == x:
        return True
    return False
```

User: Running the above code returns the following error message:
AssertionError: expect any_int(1.5, 5, 3.5) == False, but was True
So the code does not pass the assertion. Please:
1. Explain why the above code is wrong.
2. Propose the correct code.

Assistant: The wrong code does not check if all the numbers are integers. It only checks if one of the numbers is equal to the sum of the other two. This is why the assertion `any_int(1.5, 5, 3.5)` returns True, which is incorrect. Below is the fixed code.

```
def any_int(x, y, z):
    if x + y == z or x + z == y or y + z == x:
        if all(isinstance(num, int) for num in [x, y, z]):
            return True
    return False
```

*Work done while interning at AWS AI Lab

Preprint