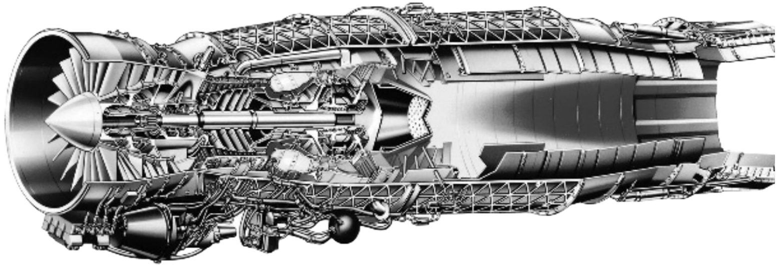


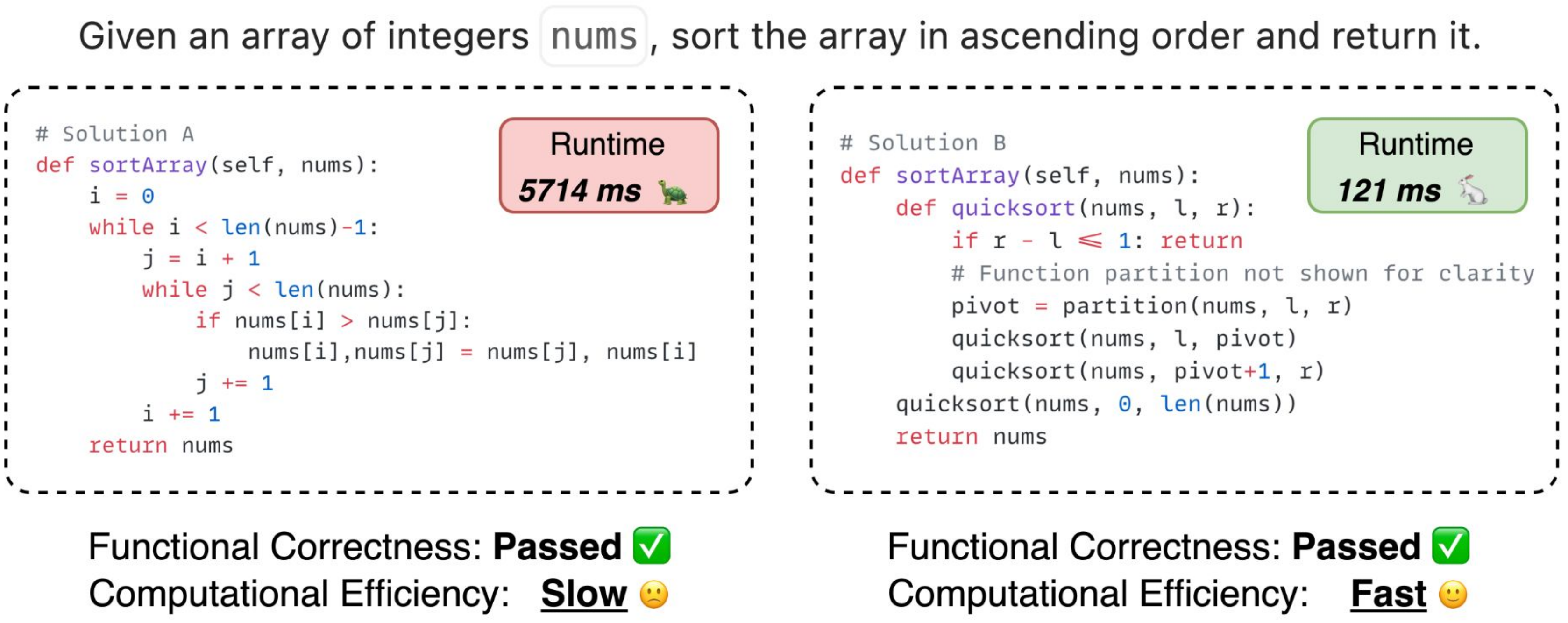
Afterburner: Reinforcement Learning Facilitates Self-Improving Code Efficiency Optimization

Mingzhe Du[1,2] **Luu Anh Tuan**[1] **Yue Liu**[2] **Yuhao Qing**[3] **Dong Huang**[2,3]†
Xinyi He[4] **Qian Liu**[5]† **Zejun Ma**[5] **See-kiong Ng**[2]

[1]Nanyang Technological University [2]National University of Singapore
[3]The University of Hong Kong [4]Xi'an Jiaotong University [5]TikTok

Outlines

1 Background & Motivation


Given an array of integers `nums`, sort the array in ascending order and return it.

```python
# Solution A
def sortArray(self, nums):
    i = 0
    while i < len(nums)-1:
        j = i + 1
        while j < len(nums):
            if nums[i] > nums[j]:
                nums[i],nums[j] = nums[j], nums[i]
            j += 1
        i += 1
    return nums
```

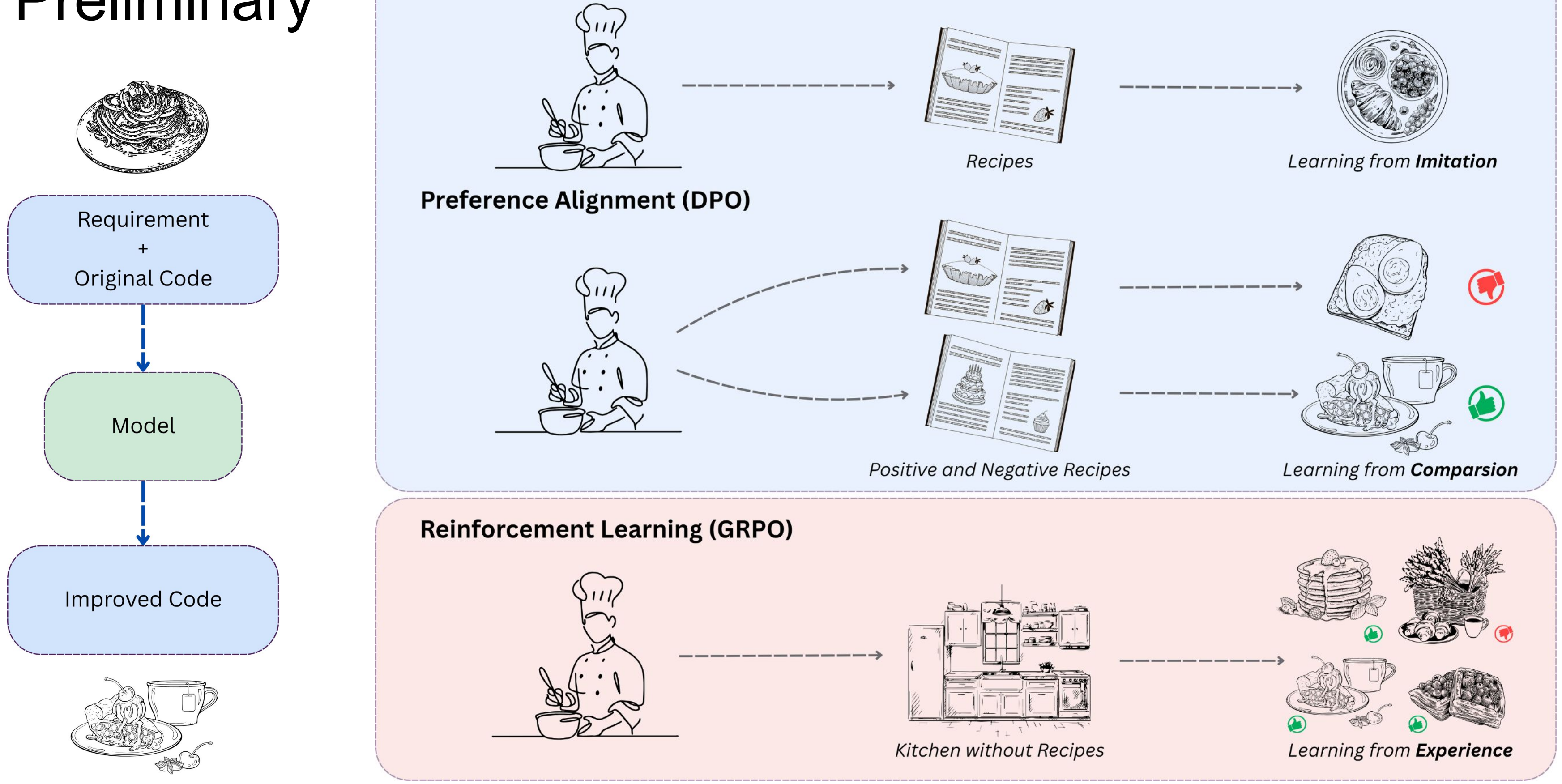
```python
# Solution B
def sortArray(self, nums):
    def quicksort(nums, l, r):
        if r - l <= 1: return
        # Function partition not shown for clarity
        pivot = partition(nums, l, r)
        quicksort(nums, l, pivot)
        quicksort(nums, pivot+1, r)
    quicksort(nums, 0, len(nums))
    return nums
```

Functional Correctness: **Passed** ✅
Computational Efficiency: **Slow** 🙁

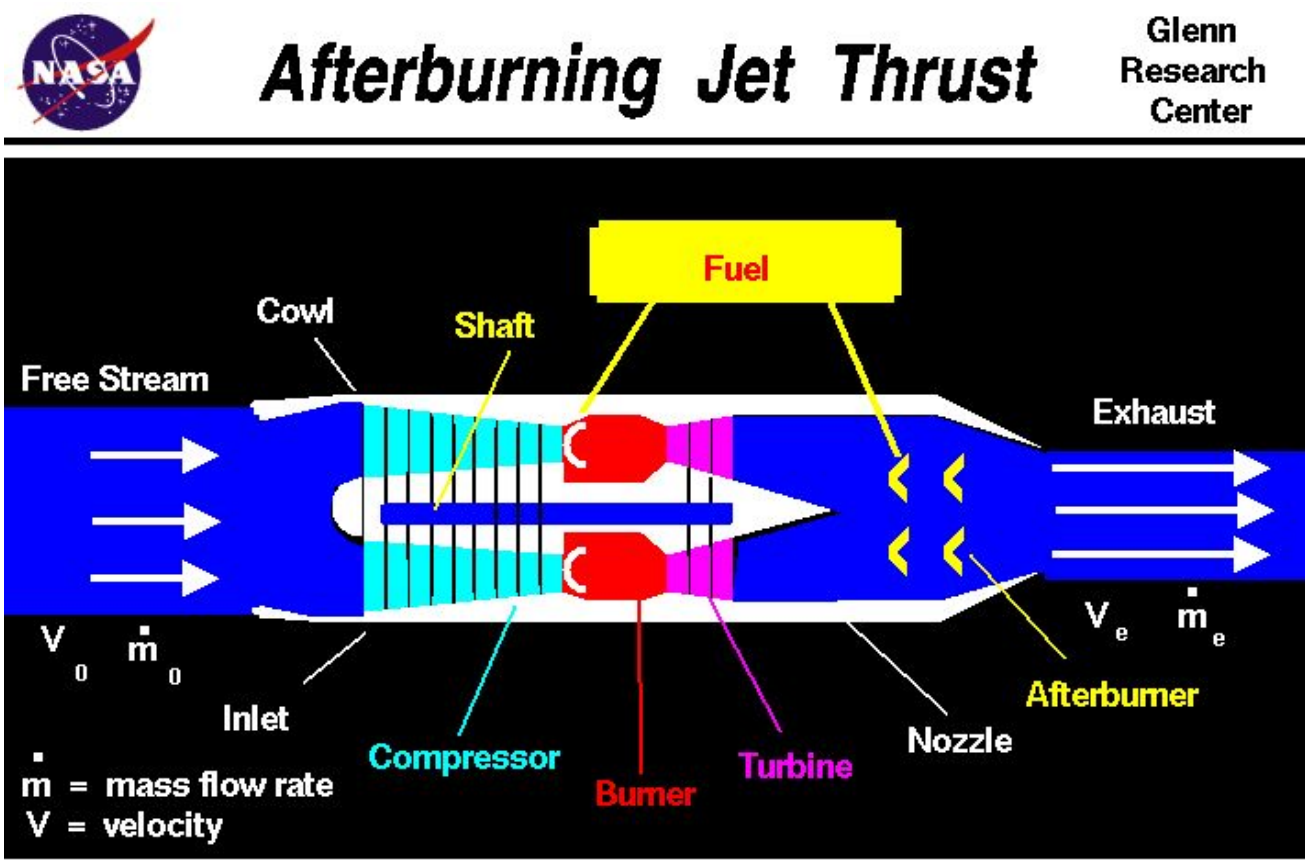
Functional Correctness: **Passed** ✅
Computational Efficiency: **Fast** 🙂

Outlines

2 Preliminary



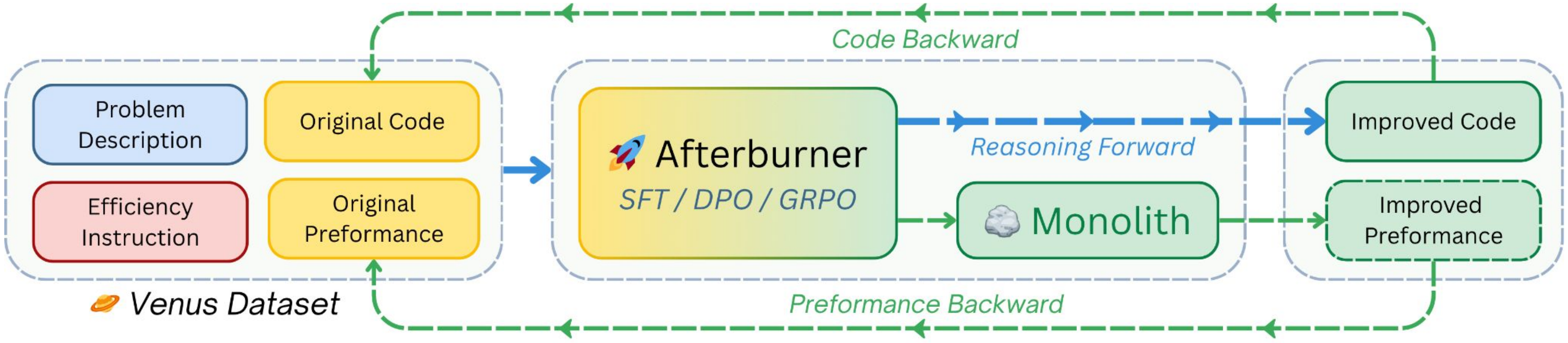
2 Preliminary



Thrust = $F = \dot{m}_e V_e - \dot{m}_0 V_0$

Outlines

3 Iterative Optimization Framework



Problem Description
Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.
You may assume that each input would have exactly one solution, and you may not use the same element twice.
You can return the answer in any order.

Efficiency Instruction
Time-efficient / Memory-efficient / Integral-efficient

Original Preformance
Passed / Time: 1788ms / Memory: 18 mb / Integral: 26192

Improved Preformance
Passed / Time: 154 ms / Memory: 20 mb / Integral: 7324

Reasoning Content

- Analyze the Original Solution:
  - *The provided solution uses a nested loop approach. <...>*
- Optimization Strategy:
  - *To improve the time complexity, we can use a hash map to store the numbers we have seen so far and their indices.*
  - *This allows us to check for the existence of the required complement in approximately O(1) average time.*
- Complexity Analysis of Optimized Solution:
  - *Time Complexity: O(n). We iterate through the list only once. Each lookup and insertion takes O(1) on average.*
  - *Space Complexity: O(n). In the worst-case scenario, the dictionary might store up to n elements if no pair is found <...>*
- Implementation Details:
  - *The Python dictionary provides the hash map functionality.*
  - *enumerate(nums) is a convenient way to iterate through the list while getting both index and value.*
  - *Ensure the output format matches the required List[int]. <...>*

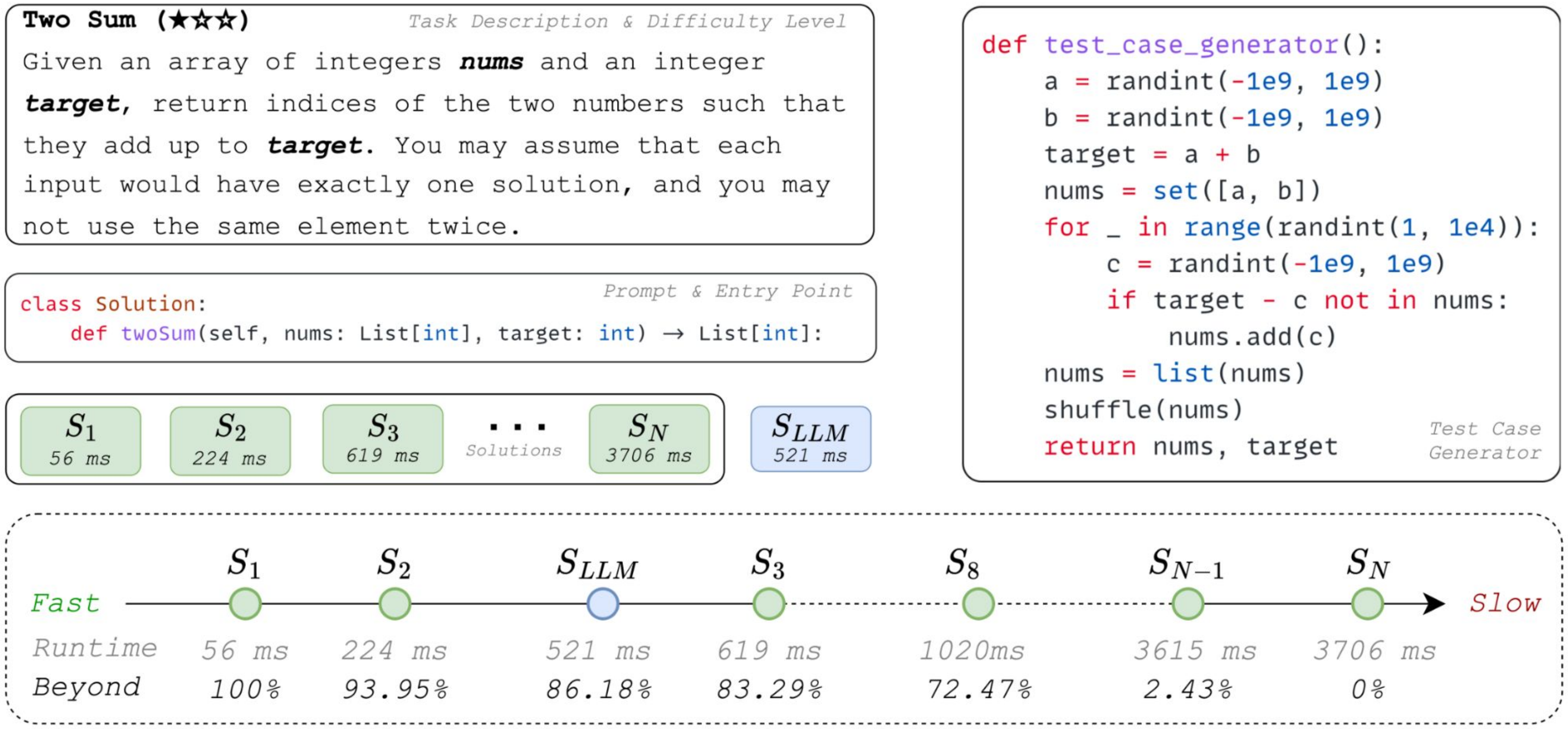
Original Code

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        i,j=0,0
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                if nums[i]+nums[j]==target:
                    return i,j
```

Improved Code

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        num_map = {}
        for i, num in enumerate(nums):
            complement = target - num
            if complement in num_map:
                return [num_map[complement], i]
            num_map[num] = i
```

3.1 Venus and Mercury (Data)



Two Sum (★☆☆) *Task Description & Difficulty Level*

Given an array of integers ***nums*** and an integer ***target***, return indices of the two numbers such that they add up to ***target***. You may assume that each input would have exactly one solution, and you may not use the same element twice.

Prompt & Entry Point

```
class Solution:
    def twoSum(self, nums: List[int], target: int) → List[int]:
```

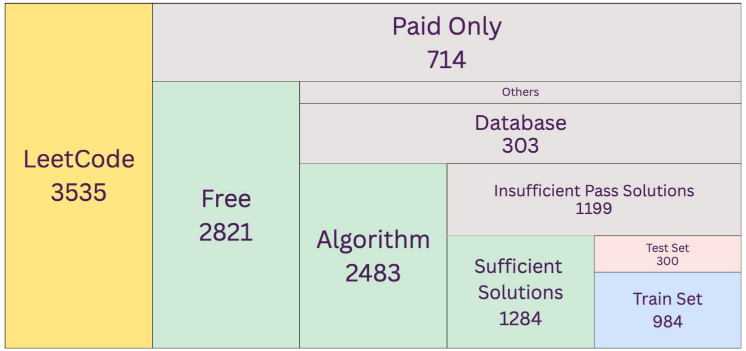
Test Case Generator

```
def test_case_generator():
    a = randint(-1e9, 1e9)
    b = randint(-1e9, 1e9)
    target = a + b
    nums = set([a, b])
    for _ in range(randint(1, 1e4)):
        c = randint(-1e9, 1e9)
        if target - c not in nums:
            nums.add(c)
    nums = list(nums)
    shuffle(nums)
    return nums, target
```

	S_1	S_2	S_{LLM}	S_3	S_8	S_{N-1}	S_N
Runtime	56 ms	224 ms	521 ms	619 ms	1020ms	3615 ms	3706 ms
Beyond	100%	93.95%	86.18%	83.29%	72.47%	2.43%	0%

3.1 Venus and Mercury (Data)

Column Name	Description
`problem_id`	Unique identifier for each problem (int64)
`title`	Title of the problem (string)
`question_content`	Full text of the problem statement (string)
`difficulty`	Difficulty level (categorical)
`tags`	List of associated tags (sequence)
`code_prompt`	Prompt used for solution generation (string)
`test_case_generator`	Code generating test cases (string)
`test_case_evaluator`	Code evaluating test case outputs (string)
`test_case_runners`	Code executing solutions with test cases (string)
`solutions`	Human-submitted solutions from LeetCode (list of strings)



Dataset	Tasks	Test Cases	Solutions	Metrics	Languages	Source
♣ HumanEval [8]	164	8.1	1.0	Pass@k	$*$ Python	Crowdsource
♣ MBPP [6]	257	3.0	1.0	Pass@k	Python	Crowdsource
♣ APPS [18]	10,000	21.2	23.4	Pass@k	$*$ Python	CodeForces
♣ BigCodeBench [66]	1,140	5.6	1.0	Pass@k	Python	Synthesis
♡ EffiBench [21]	1000	100	14.6	NET/ NMU	Python	LeetCode
♡ Mercury [14]	1,889	$+\infty$	18.4	Pass/ Beyond	Python	LeetCode
♡ ENAMEL [45]	142	20	1	Eff@k	Python	HumanEval
♡ EVALPERF [36]	1,474	–	10	DPS	Python	[8, 6, 18, 35]
♡ PIE [50]	1,889	104	80.6	%Opt / %Correct / Speedup	CPP	CodeNet
♡ ECCO [53]	48	20	16.5	Time/Memory	python	CodeNet
♡ Venus (ours)	8,598	$+\infty$	79.3	Pass/ Time/ Memory/ Integral	Multilingual	LeetCode

https://huggingface.co/datasets/Elfsong/Venus

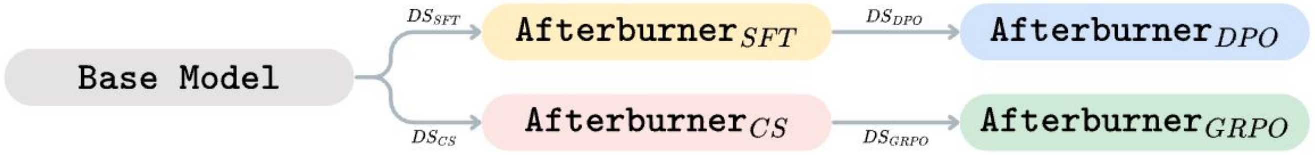
3.2 Afterburner (Model)

$$\mathcal{L}_{SFT}(\pi_\theta) = -\mathbb{E}_{(\mathcal{P},\mathcal{I},\mathcal{C}^+,\mathcal{C}^-,\mathcal{M}^-)\sim DS_{SFT}}\left[\log \pi_\theta(\mathcal{C}^+|\mathcal{X})\right],$$

$$\mathcal{L}_{DPO}(\pi_\theta;\pi_{ref}) = -\mathbb{E}_{(\mathcal{X},\mathcal{C}^+,\mathcal{C}^-)\sim DS_{DPO}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(\mathcal{C}^+|\mathcal{X})}{\pi_{ref}(\mathcal{C}^+|\mathcal{X})} - \beta \log \frac{\pi_\theta(\mathcal{C}^-|\mathcal{X})}{\pi_{ref}(\mathcal{C}^-|\mathcal{X})}\right)\right]$$

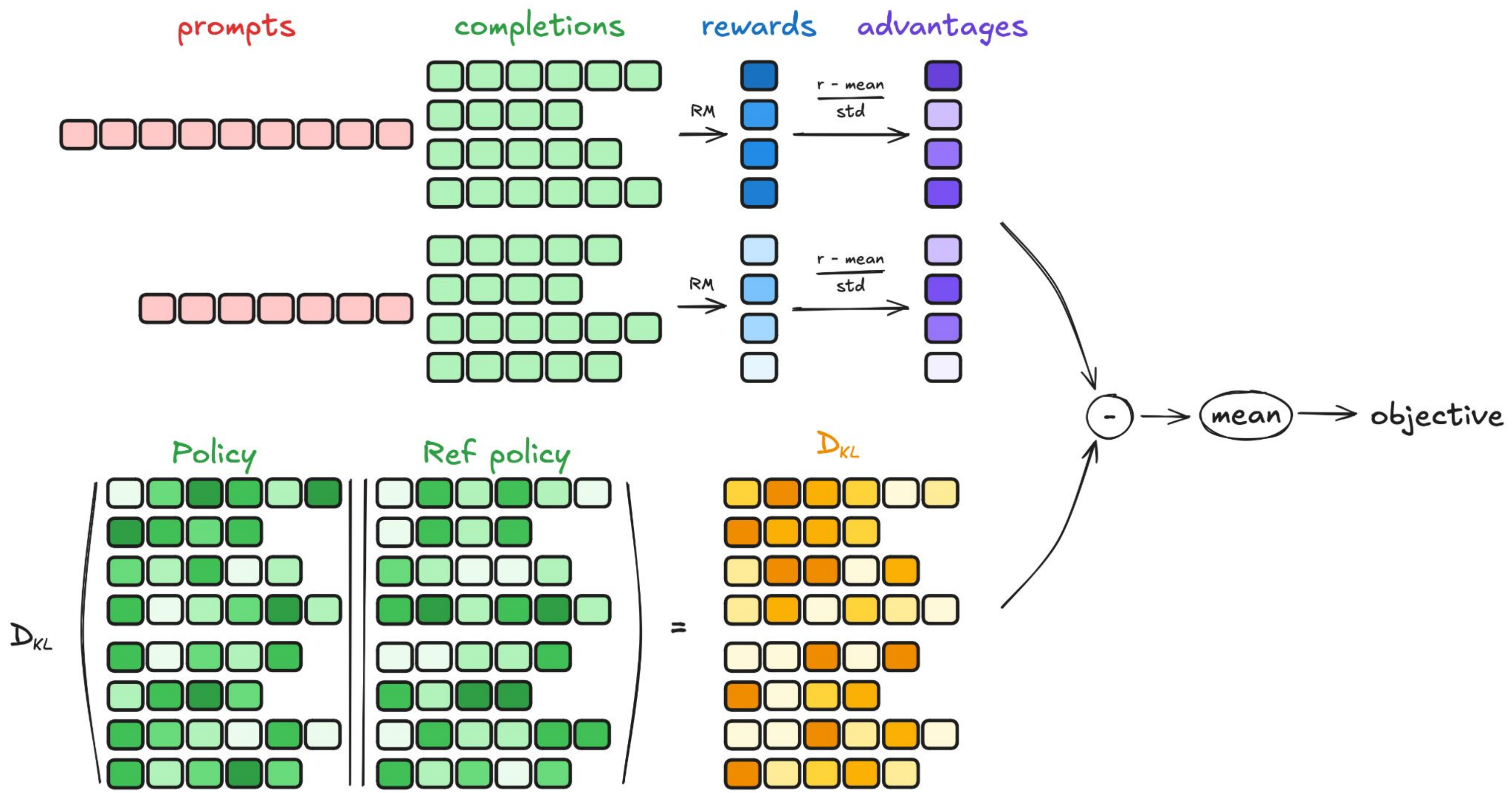
$$\mathcal{L}_{GRPO}(\pi_\theta;\pi_{\theta_{\text{old}}}) = -\mathbb{E}_{\mathcal{X}\sim DS_{GRPO},\{\mathcal{O}_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\mathcal{O}_i|\mathcal{X})}\left[\min(\mathcal{W}_i, \text{clip}(\mathcal{W}_i, 1+\epsilon, 1-\epsilon)\cdot\mathcal{A}_i)\right],$$

$$\mathcal{X} = (\mathcal{P},\mathcal{I},\mathcal{C}), \quad \mathcal{W}_i = \frac{\pi_\theta(\mathcal{O}_i|\mathcal{X})}{\pi_{\theta_{\text{old}}}(\mathcal{O}_i|\mathcal{X})}, \quad \mathcal{A}_i = \frac{\mathcal{R}_i - \text{mean}(\{\mathcal{R}_i\}_{i=1}^{G})}{\text{std}(\{\mathcal{R}_i\}_{i=1}^{G})},$$



https://github.com/Elfsong/Afterburner

3.2 Afterburner (Model)



3.2 Monolith (Critic)

$$\{passed, time, memory, integral\} = \texttt{Monolith}(code, test\_cases)$$

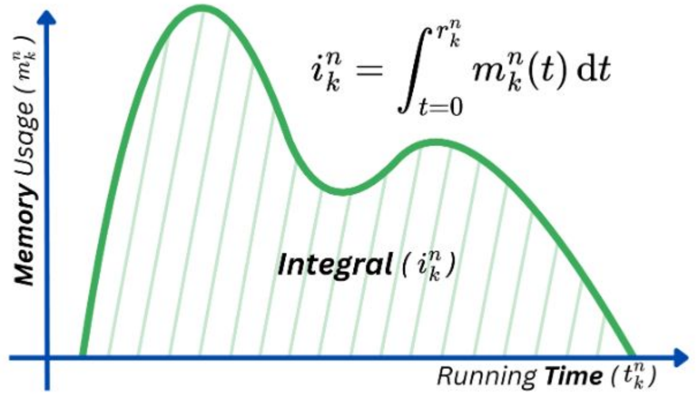


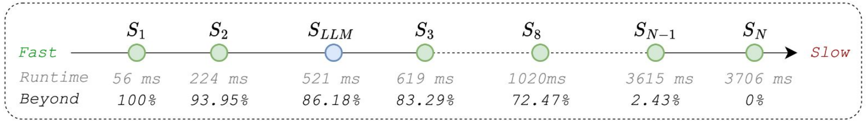
Figure 3: Illustration of task-level efficiency metrics.

- *Functional Correctness:*

$$\text{Pass@1} = \mathcal{N}_{passed}/\mathcal{N}_{total}$$

- *Code Efficiency:*

$$\text{Beyond-}\{\text{T, M, I}\} = \frac{\sum_{k=1}^{|V|} \text{PR}(\mathcal{E}_k^{gen}, \{D_k^T, D_k^M, D_k^I\})}{|V_{test}|}$$



	S_1	S_2	S_{LLM}	S_3	S_8	S_{N-1}	S_N
Runtime	56 ms	224 ms	521 ms	619 ms	1020ms	3615 ms	3706 ms
Beyond	100%	93.95%	86.18%	83.29%	72.47%	2.43%	0%

https://github.com/Elfsong/Monolith

3.2 Monolith (Critic)

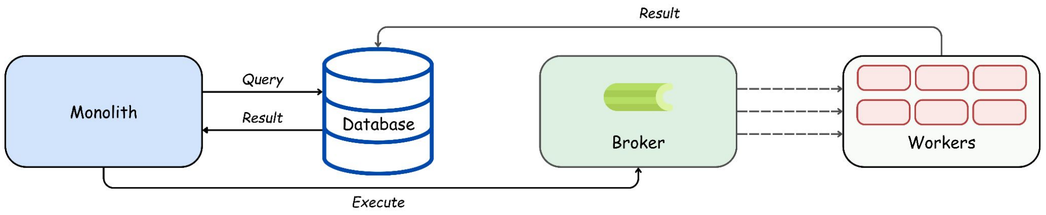
$$R_{correct}(C^{in}, C^{out}) = \begin{cases} 1.0 & \text{if } \mathcal{A}^{out} = 1 \text{ and } \mathcal{A}^{in} = 0 \text{ (upgrade)} \\ 0.5 & \text{if } \mathcal{A}^{out} = 1 \text{ and } \mathcal{A}^{in} = 1 \text{ (maintained passing status)} \\ -0.5 & \text{if } \mathcal{A}^{out} = 0 \text{ and } \mathcal{A}^{in} = 0 \text{ (maintained failing status)} \\ -1.0 & \text{if } \mathcal{A}^{out} = 0 \text{ and } \mathcal{A}^{in} = 1 \text{ (downgrade)} \end{cases}$$

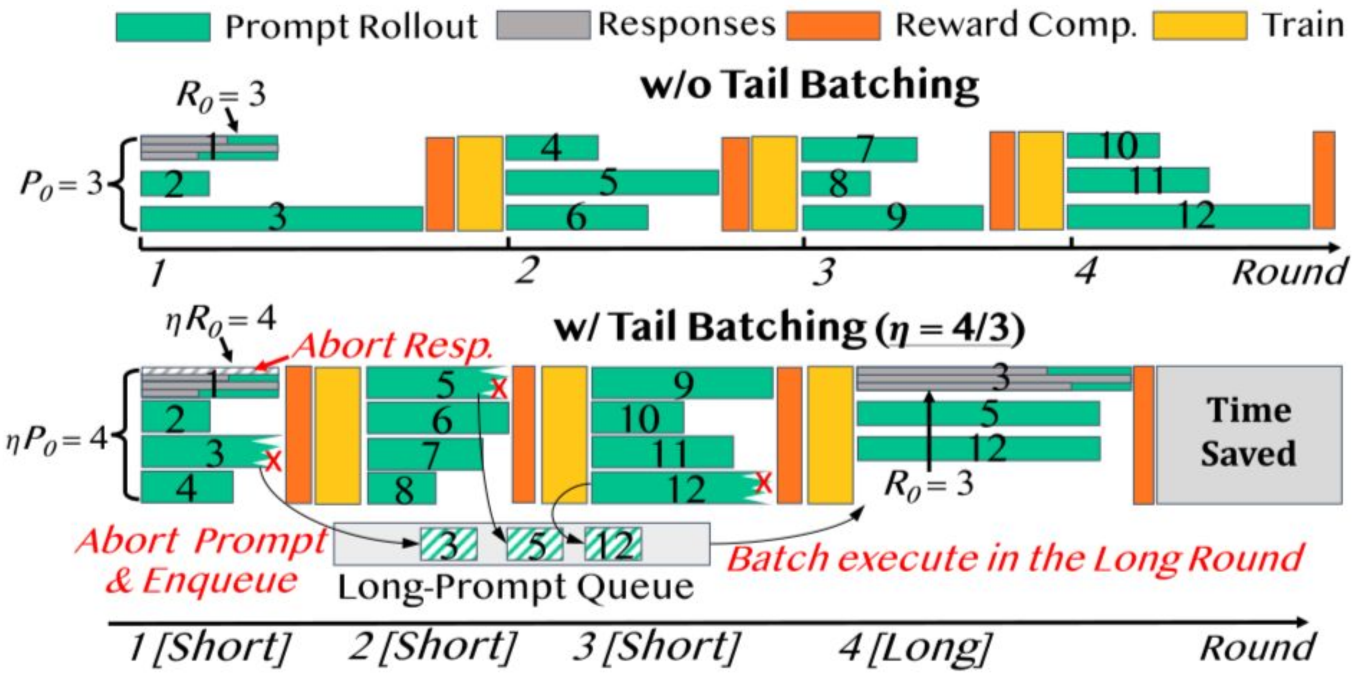
$$\mathcal{R}_{efficiency} = \tanh(\mathcal{E}_{gain}), \quad \mathcal{E}_{\text{gain}} = \frac{\mathcal{E}_{clip}^{\,in} - \mathcal{E}_{clip}^{\,out}}{\mathcal{E}_{clip}^{\,in} + \epsilon}, \quad \mathcal{E}_{clip} = \text{clip}(\mathcal{E}, 0, \mathcal{E}_{upper}),$$

$$\mathcal{R}_{final} = \beta_f \cdot \mathcal{R}_{format} + \beta_c \cdot \mathcal{R}_{correct} + \beta_e \cdot \mathcal{R}_{efficiency}$$

https://github.com/Elfsong/Monolith

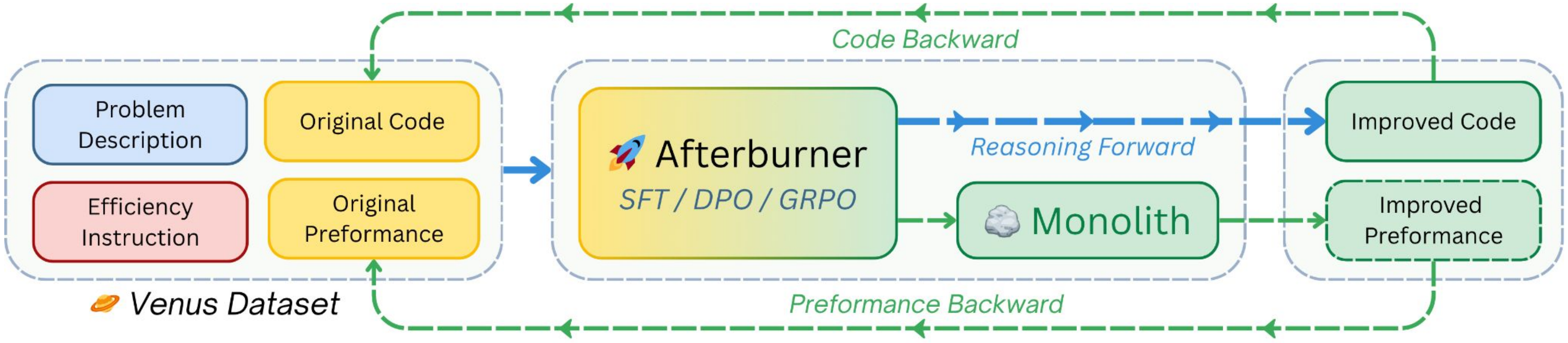
3.2 Monolith (Critic)





RollPacker: Mitigating Long-Tail Rollouts for Fast, Synchronous RL Post-Training

3 Iterative Optimization Framework



Problem Description
Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.
You may assume that each input would have exactly one solution, and you may not use the same element twice.
You can return the answer in any order.

Efficiency Instruction
Time-efficient / Memory-efficient / Integral-efficient

Original Preformance
Passed / Time: 1788ms / Memory: 18 mb / Integral: 26192

Improved Preformance
Passed / Time: 154 ms / Memory: 20 mb / Integral: 7324

Reasoning Content

- Analyze the Original Solution:
  - *The provided solution uses a nested loop approach. <...>*
- Optimization Strategy:
  - *To improve the time complexity, we can use a hash map to store the numbers we have seen so far and their indices.*
  - *This allows us to check for the existence of the required complement in approximately O(1) average time.*
- Complexity Analysis of Optimized Solution:
  - *Time Complexity: O(n). We iterate through the list only once. Each lookup and insertion takes O(1) on average.*
  - *Space Complexity: O(n). In the worst-case scenario, the dictionary might store up to n elements if no pair is found <...>*
- Implementation Details:
  - *The Python dictionary provides the hash map functionality.*
  - *enumerate(nums) is a convenient way to iterate through the list while getting both index and value.*
  - *Ensure the output format matches the required List[int]. <...>*

Original Code

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        i,j=0,0
        for i in range(len(nums)):
            for j in range(i+1,len(nums)):
                if nums[i]+nums[j]==target:
                    return i,j
```

Improved Code

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        num_map = {}
        for i, num in enumerate(nums):
            complement = target - num
            if complement in num_map:
                return [num_map[complement], i]
            num_map[num] = i
```

Outlines

4.1 How about the Code Efficiency Performance of Vanilla LLMs?

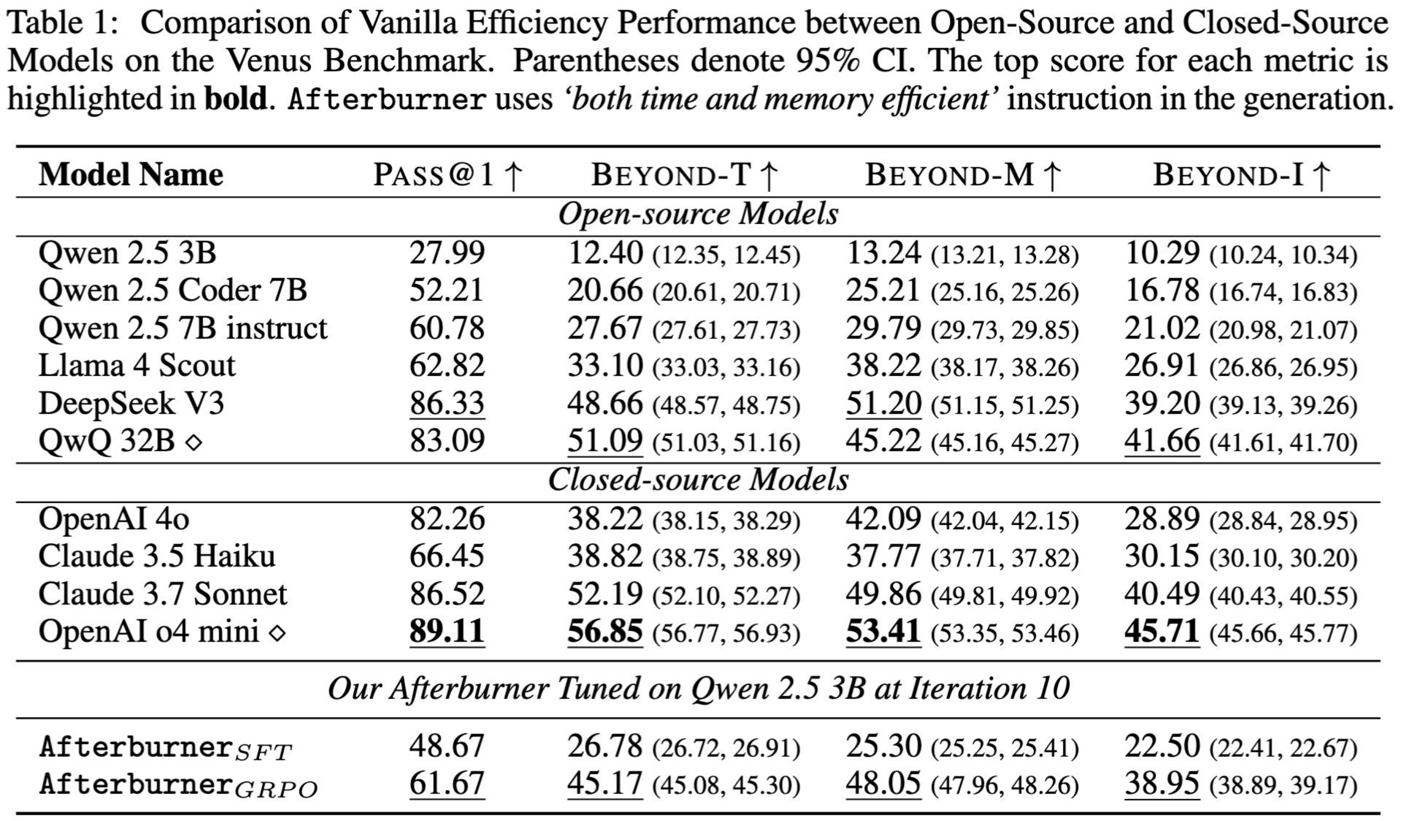


Table 1: Comparison of Vanilla Efficiency Performance between Open-Source and Closed-Source Models on the Venus Benchmark. Parentheses denote 95% CI. The top score for each metric is highlighted in **bold**. `Afterburner` uses *'both time and memory efficient'* instruction in the generation.

Model Name	PASS@1 ↑	BEYOND-T ↑	BEYOND-M ↑	BEYOND-I ↑
		Open-source Models		
Qwen 2.5 3B	27.99	12.40 (12.35, 12.45)	13.24 (13.21, 13.28)	10.29 (10.24, 10.34)
Qwen 2.5 Coder 7B	52.21	20.66 (20.61, 20.71)	25.21 (25.16, 25.26)	16.78 (16.74, 16.83)
Qwen 2.5 7B instruct	60.78	27.67 (27.61, 27.73)	29.79 (29.73, 29.85)	21.02 (20.98, 21.07)
Llama 4 Scout	62.82	33.10 (33.03, 33.16)	38.22 (38.17, 38.26)	26.91 (26.86, 26.95)
DeepSeek V3	86.33	48.66 (48.57, 48.75)	51.20 (51.15, 51.25)	39.20 (39.13, 39.26)
QwQ 32B ◇	83.09	51.09 (51.03, 51.16)	45.22 (45.16, 45.27)	41.66 (41.61, 41.70)
		Closed-source Models		
OpenAI 4o	82.26	38.22 (38.15, 38.29)	42.09 (42.04, 42.15)	28.89 (28.84, 28.95)
Claude 3.5 Haiku	66.45	38.82 (38.75, 38.89)	37.77 (37.71, 37.82)	30.15 (30.10, 30.20)
Claude 3.7 Sonnet	86.52	52.19 (52.10, 52.27)	49.86 (49.81, 49.92)	40.49 (40.43, 40.55)
OpenAI o4 mini ◇	**89.11**	**56.85** (56.77, 56.93)	**53.41** (53.35, 53.46)	**45.71** (45.66, 45.77)
		Our Afterburner Tuned on Qwen 2.5 3B at Iteration 10		
`Afterburner`$_{SFT}$	48.67	26.78 (26.72, 26.91)	25.30 (25.25, 25.41)	22.50 (22.41, 22.67)
`Afterburner`$_{GRPO}$	61.67	45.17 (45.08, 45.30)	48.05 (47.96, 48.26)	38.95 (38.89, 39.17)

4.2 Does Iterative Improvement Framework Work?

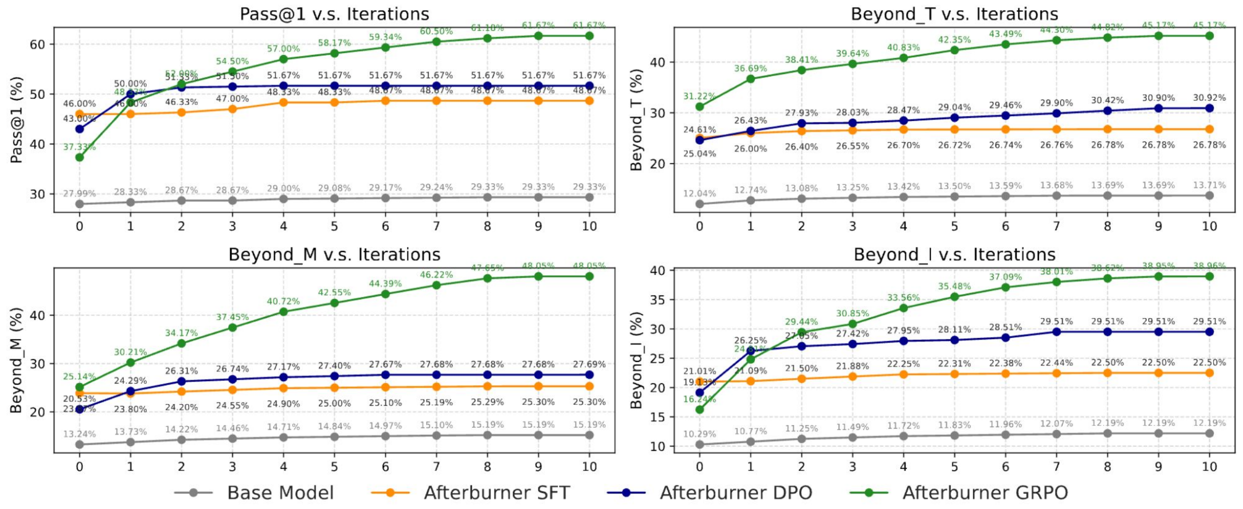
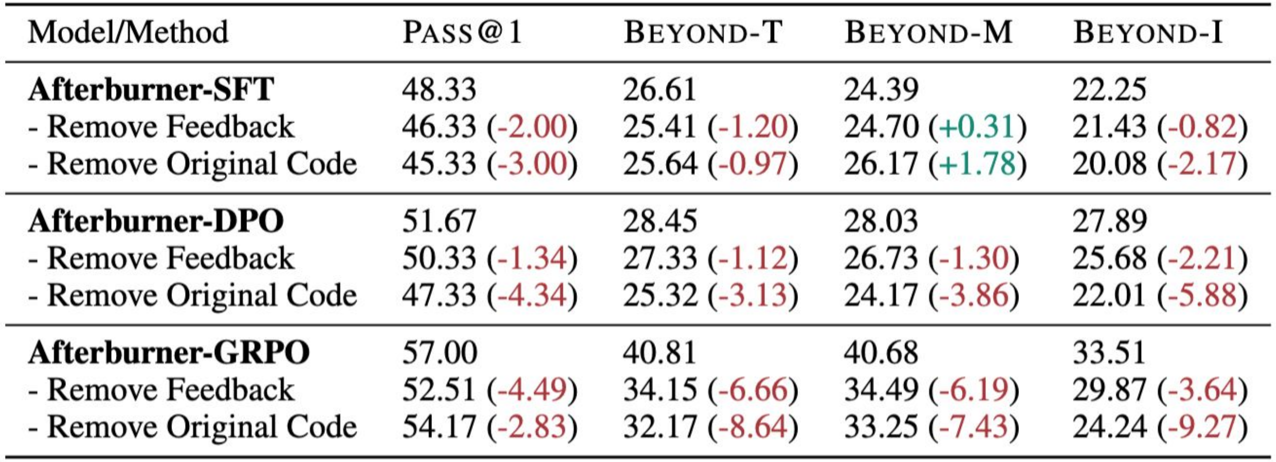


Figure 5: Iterative Optimization with an Efficient Instruction *'both time and memory efficient'*.

- SFT **Memorized** Superficial Patterns;
- DPO **Realized** Static Preferences;
- GRPO **Cultivated** Adaptive Proficiency.

4.3 Why GRPO Can Iteratively Enhance Code Efficiency?



Model/Method	PASS@1	BEYOND-T	BEYOND-M	BEYOND-I
Afterburner-SFT	48.33	26.61	24.39	22.25
- Remove Feedback	46.33 (-2.00)	25.41 (-1.20)	24.70 (+0.31)	21.43 (-0.82)
- Remove Original Code	45.33 (-3.00)	25.64 (-0.97)	26.17 (+1.78)	20.08 (-2.17)
Afterburner-DPO	51.67	28.45	28.03	27.89
- Remove Feedback	50.33 (-1.34)	27.33 (-1.12)	26.73 (-1.30)	25.68 (-2.21)
- Remove Original Code	47.33 (-4.34)	25.32 (-3.13)	24.17 (-3.86)	22.01 (-5.88)
Afterburner-GRPO	57.00	40.81	40.68	33.51
- Remove Feedback	52.51 (-4.49)	34.15 (-6.66)	34.49 (-6.19)	29.87 (-3.64)
- Remove Original Code	54.17 (-2.83)	32.17 (-8.64)	33.25 (-7.43)	24.24 (-9.27)

- *Generation **diversity** is foundational to its iterative capability.*
- *GRPO gains **experience** improving code from what it generated through the iterative refinement loop.*

4.4 Can Afterburner Generate Code Surpassing Human Efficiency?

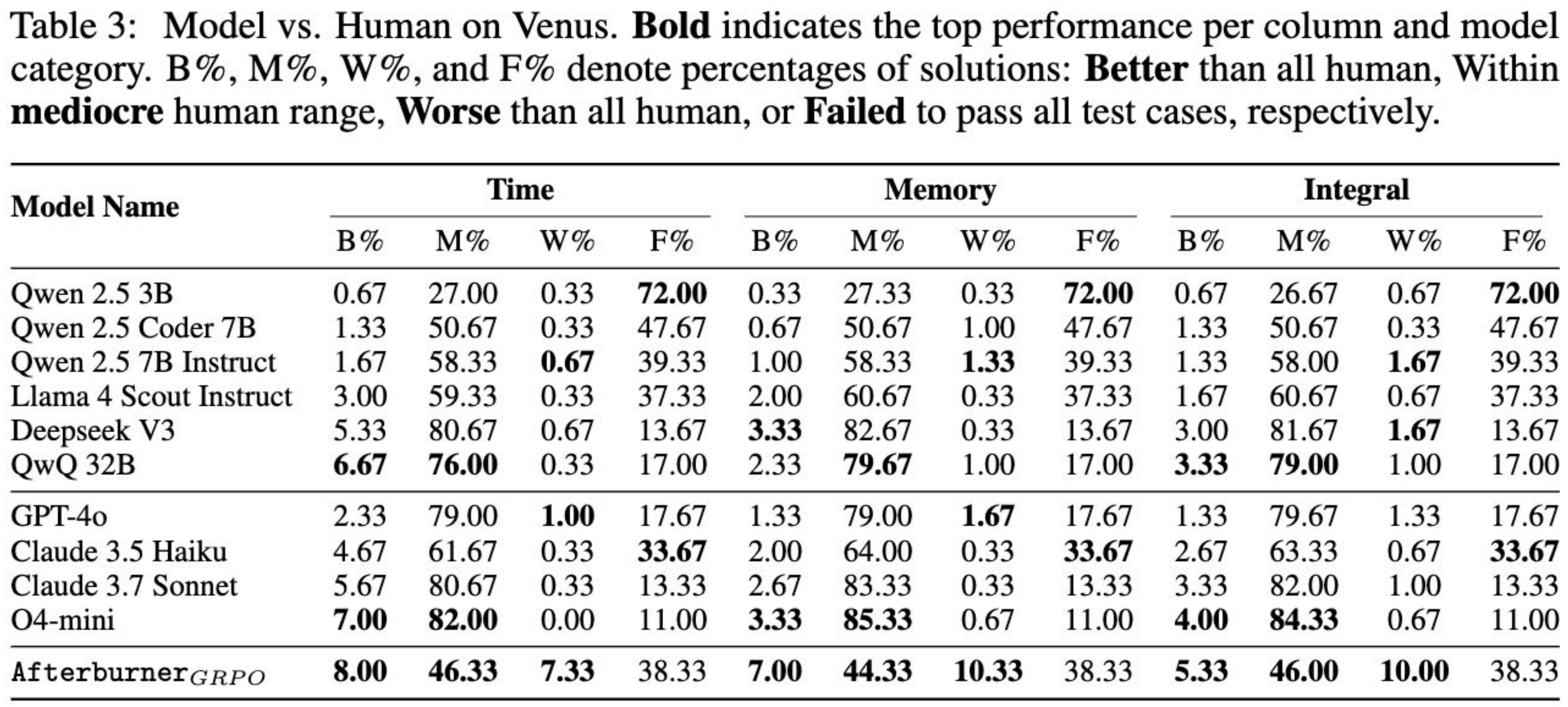


Table 3: Model vs. Human on Venus. **Bold** indicates the top performance per column and model category. B%, M%, W%, and F% denote percentages of solutions: **Better** than all human, Within **mediocre** human range, **Worse** than all human, or **Failed** to pass all test cases, respectively.

Model Name	Time				Memory				Integral			
	B%	M%	W%	F%	B%	M%	W%	F%	B%	M%	W%	F%
Qwen 2.5 3B	0.67	27.00	0.33	**72.00**	0.33	27.33	0.33	**72.00**	0.67	26.67	0.67	**72.00**
Qwen 2.5 Coder 7B	1.33	50.67	0.33	47.67	0.67	50.67	1.00	47.67	1.33	50.67	0.33	47.67
Qwen 2.5 7B Instruct	1.67	58.33	**0.67**	39.33	1.00	58.33	**1.33**	39.33	1.33	58.00	**1.67**	39.33
Llama 4 Scout Instruct	3.00	59.33	0.33	37.33	2.00	60.67	0.33	37.33	1.67	60.67	0.67	37.33
Deepseek V3	5.33	80.67	0.67	13.67	**3.33**	82.67	0.33	13.67	3.00	81.67	**1.67**	13.67
QwQ 32B	**6.67**	**76.00**	0.33	17.00	2.33	**79.67**	1.00	17.00	**3.33**	**79.00**	1.00	17.00
GPT-4o	2.33	79.00	**1.00**	17.67	1.33	79.00	**1.67**	17.67	1.33	79.67	1.33	17.67
Claude 3.5 Haiku	4.67	61.67	0.33	**33.67**	2.00	64.00	0.33	**33.67**	2.67	63.33	0.67	**33.67**
Claude 3.7 Sonnet	5.67	80.67	0.33	13.33	2.67	83.33	0.33	13.33	3.33	82.00	1.00	13.33
O4-mini	**7.00**	**82.00**	0.00	11.00	**3.33**	**85.33**	0.67	11.00	**4.00**	**84.33**	0.67	11.00
Afterburner$_{GRPO}$	**8.00**	**46.33**	**7.33**	38.33	**7.00**	**44.33**	**10.33**	38.33	**5.33**	**46.00**	**10.00**	38.33

Thank you!