Towards Efficient Code Generation

from evaluation to generation

Du Mingzhe

Nanyang Technological University (NTU)

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
```

Runtime
5714 ms 🐢

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

```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
```

Runtime
121 ms 🐇

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

Towards Efficient Code Generation

Measurement

Infrastructure

Phase 1: The Yardstick

(Mercury, NeurIPS'24)

Measuring the Efficiency Gap

- Precise Benchmarking
- Efficiency Metrics

Phase 2: The Forge

(CodeArena, ACL'25)

Scalable Training & Evaluation

- Unified Infrastructure
- Massive Scale

Phase 3: The Engine

(Afterburner, NeurIPS'25)

Self-Improving Code via RL

- Reinforcement Learning
- Iterative Optimization

Mercury: A Code Efficiency Benchmark

- **Problem:** Existing benchmarks focus on correctness, neglecting efficiency.
- **Solution:** First efficiency benchmark with 1,889 Python tasks.
- **Metric:** 'Beyond' (runtime-percentile-weighted Pass score).

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
```

Runtime 5714 ms

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

```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
```

Runtime 121 ms

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

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

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

S_1 56 ms | S_2 224 ms | S_3 619 ms | ... Solutions | S_N 3706 ms | S_{LLM} 521 ms

```python
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
```

Test Case Generator

	S_1	S_2	S_{LLM}	S_3	S_8	S_{N-1}	S_N
Fast → Slow							
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%

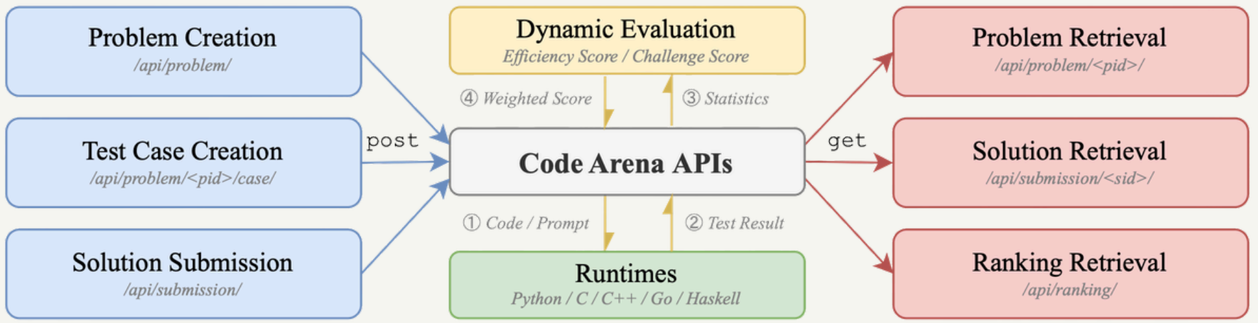
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Du Mingzhe

CodeArena: Scalable Training & Evaluation Forge

- **Problem:** Benchmark Contamination, Data Dissipation, and Poor Platform Accessibility.
- **Solution:** An online OJ platform with Dynamic Points (anti-leakage) and Open Data Repository.

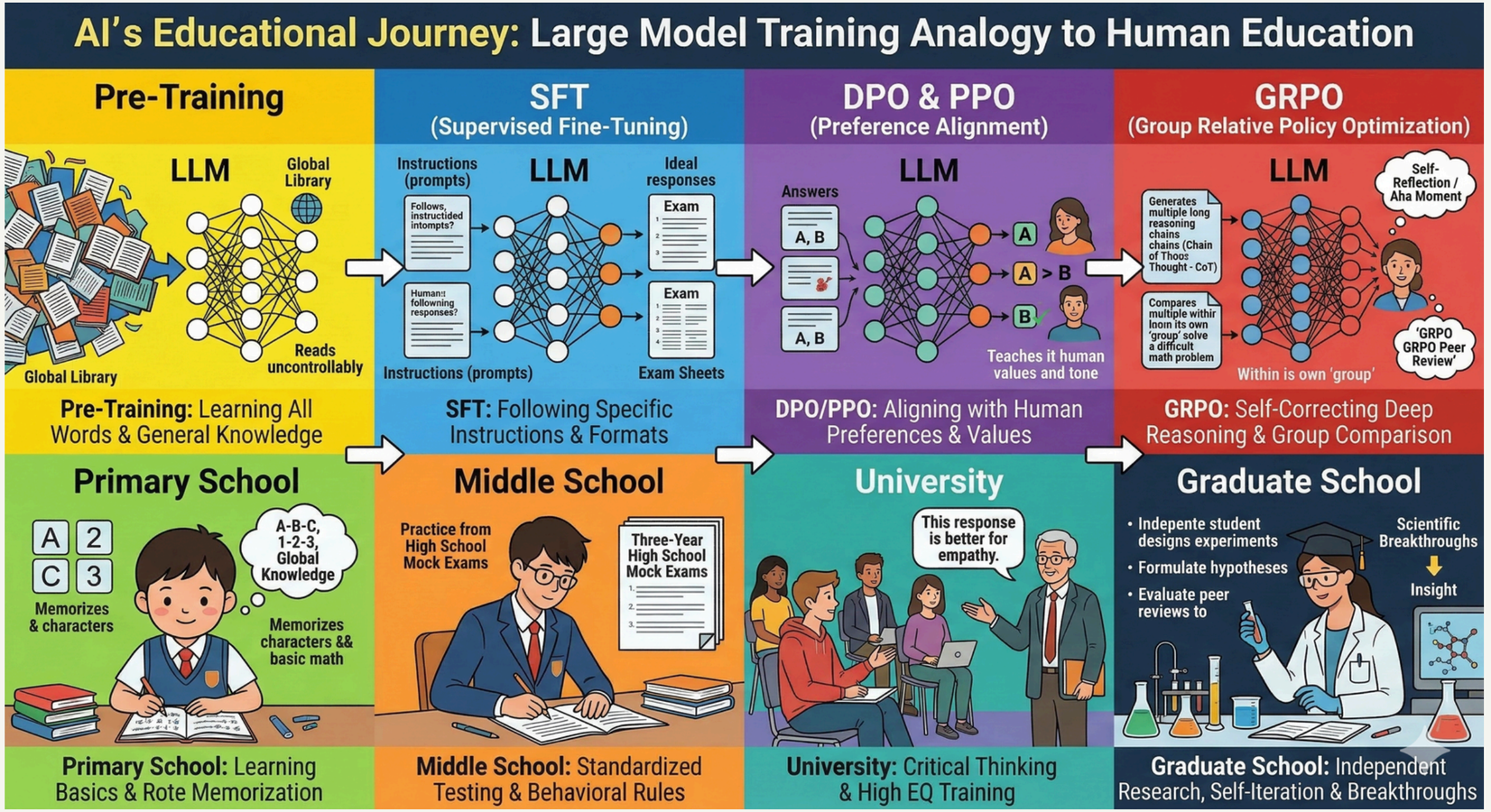
Models / Problems	Model 1	Model 2	Model 3
Problem 1 (5 pts)	✗ $\mathcal{CS}=0, \mathcal{ES}=0$	✓ 58 ms $\mathcal{CS}=5, \mathcal{ES}=0.8$	✗ $\mathcal{CS}=0, \mathcal{ES}=0$
Problem 2 (3 pts)	✓ 42 ms $\mathcal{CS}=1, \mathcal{ES}=0.4$	✓ 24 ms $\mathcal{CS}=1, \mathcal{ES}=0.6$	✓ 48 ms $\mathcal{CS}=1, \mathcal{ES}=0.3$
Problem 3 (5 pts)	✗ $\mathcal{CS}=0, \mathcal{ES}=0$	✓ 19 ms $\mathcal{CS}=2.5, \mathcal{ES}=0.5$	✓ 22 ms $\mathcal{CS}=2.5, \mathcal{ES}=0.4$
Statistics	$\mathcal{DP}=0+1.4+0=\mathbf{1.4}$	$\mathcal{DP}=5.8+1.6+3=\mathbf{10.4}$	$\mathcal{DP}=0+1.3+2.9=\mathbf{4.2}$



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

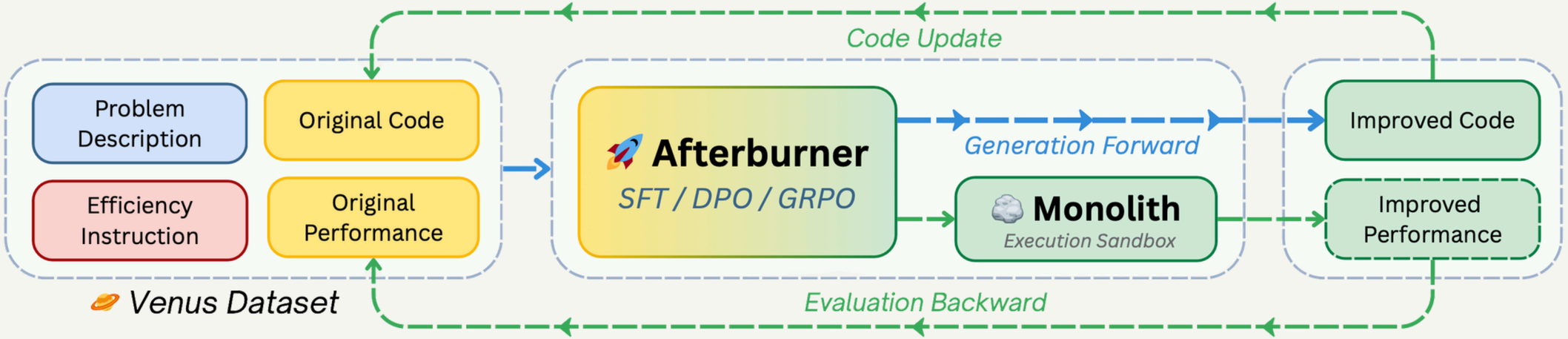
Du Mingzhe

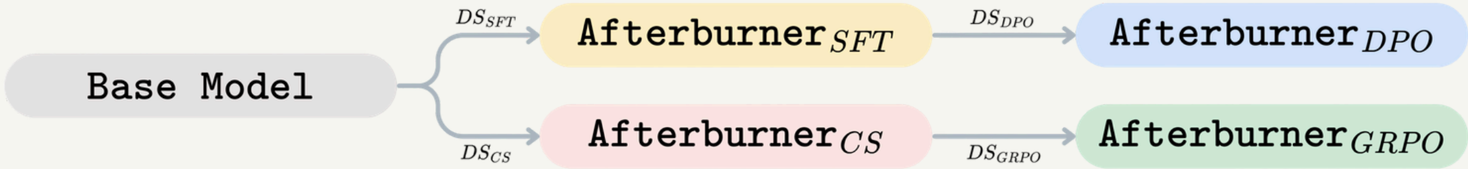
Afterburner: RL for Self-Improving Efficiency



Afterburner: RL for Self-Improving Efficiency

- **Problem:** SFT and DPO plateau in efficiency gains; models don't learn to optimize.
- **Solution:** A test-time iterative optimization framework using Reinforcement Learning.





NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Du Mingzhe

Future Work

- **Repo-level Code Efficiency**
  - **"Swe-perf: Can language models optimize code performance on real-world repositories?"** *He, Xinyi, Qian Liu, **Mingzhe Du**, Lin Yan, Zhijie Fan, Yiming Huang, Zejian Yuan, and Zejun Ma. "arXiv preprint arXiv:2507.12415 (2025).*
  - **"Benchmarking llms for unit test generation from real-world functions."** *Huang, Dong, Jie M. Zhang, Mark Harman, Qianru Zhang, **Mingzhe Du**, and See-Kiong Ng.arXiv preprint arXiv:2508.00408 (2025).*

- **Multi-lingual Efficiency Benchmark**
  - **"Effibench-x: A multi-language benchmark for measuring efficiency of llm-generated code."** *Qing, Yuhao, Boyu Zhu, **Mingzhe Du**, Zhijiang Guo, Terry Yue Zhuo, Qianru Zhang, Jie M. Zhang et al. arXiv preprint arXiv:2505.13004 (2025).*

- **Multi-agent Coding**
  - **"Nexus: Execution-Grounded Multi-Agent Test Oracle Synthesis."** *Huang, Dong, **Mingzhe Du**, Jie M. Zhang, Zheng Lin, Meng Luo, Qianru Zhang, and See-Kiong Ng. arXiv preprint arXiv:2510.26423 (2025).*

References

[1] Du, M., Luu, A. T., Ji, B., Liu, Q., & Ng, S. K. (2024). Mercury: A code efficiency benchmark for code large language models. Advances in Neural Information Processing Systems, 37, 16601-16622.

[2] Du, M., Tuan, L. A., Ji, B., Wu, X., Qing, Y., Huang, D., ... & Ng, S. K. (2025, July). Codearena: A collective evaluation platform for llm code generation. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations) (pp. 502-512).

[3] Du, M., Tuan, L. A., Liu, Y., Qing, Y., Huang, D., He, X., ... & Ng, S. K. (2025). Afterburner: Reinforcement learning facilitates self-improving code efficiency optimization. arXiv preprint arXiv:2505.23387.