Chain of Execution Supervision Promotes General Reasoning in Large Language Models

Presenter

Nuo Chen

Qwen Team, Alibaba Group

Other Authors

Zehua Li, Keqin Bao, Junyang Lin, Dayiheng Liu

Motivation & Contribution

- The Challenge: General Reasoning is the Bottleneck
  - LLM reasoning capabilities are often siloed, failing to generalize across domains.
  - Raw code is a rich source of logic, but its reasoning signals are implicit, noisy, and entangled, making direct training suboptimal.

We argue that converting each step of code execution into a CoT-style natural language narration would produce data that is not only logically rigorous and well-structured but also offers clearer, more interpretable reasoning traces.

Motivation & Contribution

- Our Solution: Chain of Execution (CoE) Supervision
  - We introduce Chain of Execution (CoE), a novel method to distill implicit code execution into explicit, step-by-step natural language rationales.
  - We built TracePile: a massive 2.6 Million sample corpus of CoE traces, spanning mathematics, classical algorithms, and algorithmic competition code.
  - Our data is enriched with granular variable-tracing questions and diverse code rewritings to promote deeper logical understanding

Core Concept: Chain of Execution (CoE)

- Turns code execution into explicit natural-language rationales.
- Enables models to learn procedural reasoning instead of memorizing I/O pairs.
- Example: Trace variable states, predict intermediate steps, and explain execution flow.

Overview

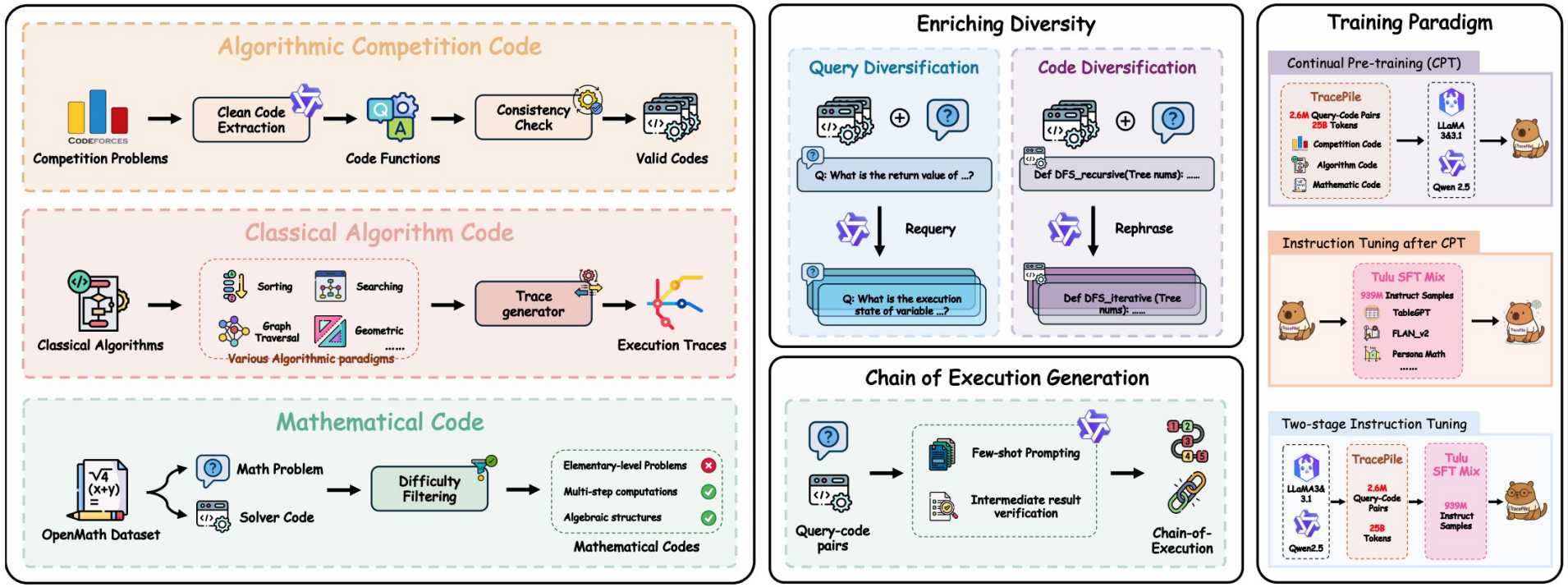

Figure 1: The curation process of TracePile and training pipelines. The left part indicates the sources of Tracepile include algorithmic competition, classical algorithm and mathematical code. The middle part shows the strategies to enrich the data diversity and CoE-style data generation. The right part includes the three different training paradigm in this work.

Building TracePile

- A 2.6M-sample corpus for reasoning supervision:
- Sources:
- - Algorithmic competition (Codeforces)
- - Classical algorithms (CLRS-style)
- - Mathematical code (OpenMath)
- Diversity augmentation:
- - Query diversification (variable tracing)
- - Code diversification (structure-preserving rewrites).

Generation Pipeline

- Data collection & filtering
- Few-shot prompting with Qwen-2.5-72B
- Execution verification with ground-truth traces
- Quality control (multi-path sampling, 8k-token limit).

Table 2: Statistics of TracePile.

Components	**Size**	**Tokens**
Algorithmic Competition	480,782	3,891,991,488
Classical Algorithm	949,088	7,024,210,656
Mathematical	1,170,836	7,105,488,512
Total	**2,600,706**	**19,380,461,000**

CoE Cases

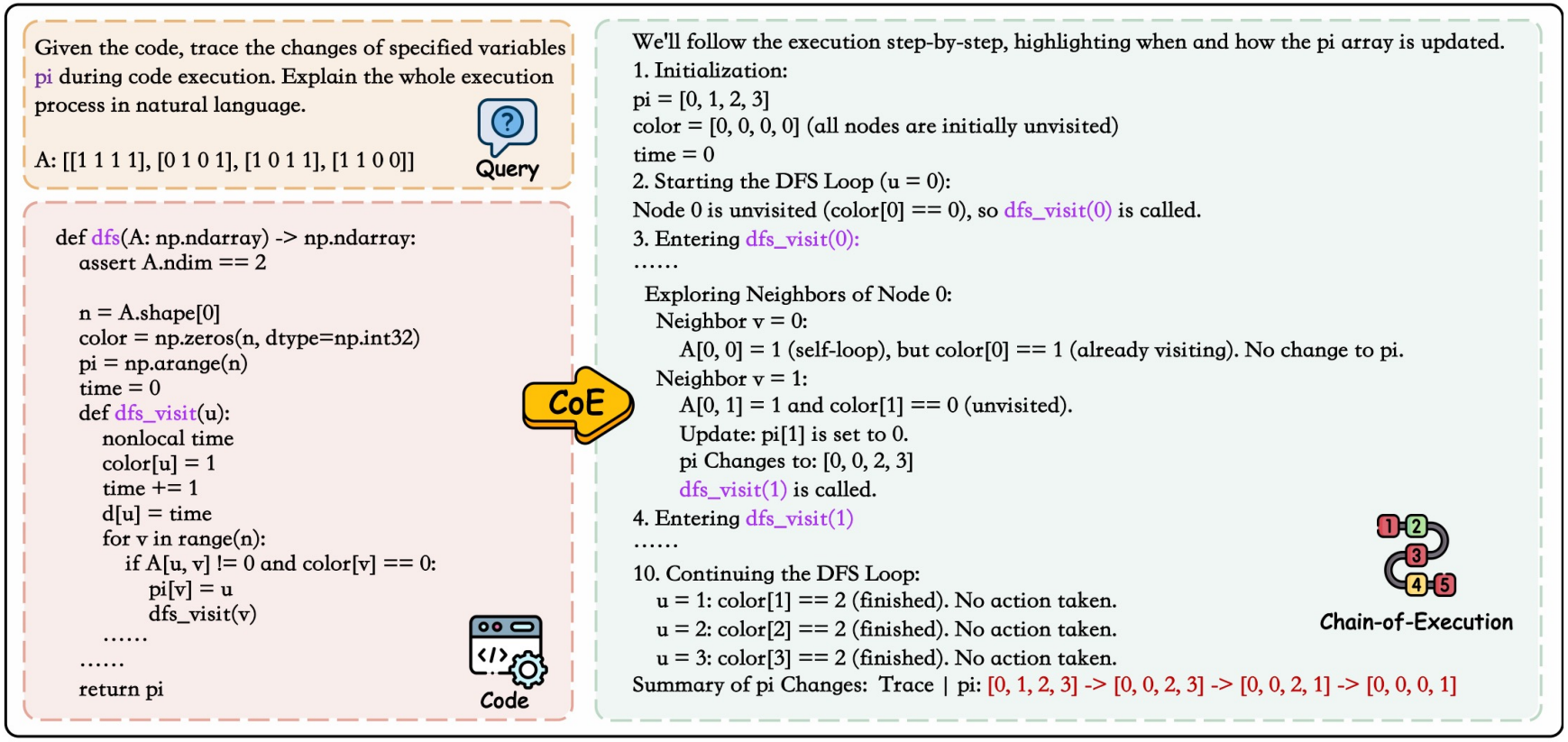

Figure 2: A classical DFS algorithm example of CoE in TracePile. More cases are in Appendix C.

Training Paradigm

- 1. Continue-pretraining on TracePile;
- 2. Instruction tuning after pretraining;
- 3. Two-stage instruction-tuning.

Model performances under continue-pretraining

Table 1: Model performances under continue-pretraining. L.C.Bench-O refers to the test-outpu subset of LiveCodeBench. The best results are in purple.

Type	Datasets	Llama-3-8B		Llama-3.1-8B		Qwen-2.5-7B		Qwen-2.5-Coder	
		Base	Ours	Base	Ours	Base	Ours	Base	Ours
Math	GSM8K	54.2	**74.3**	54.4	**74.7**	**85.7**	84.9	77.9	**83.5**
	MATH	16.5	**31.7**	17.7	**29.3**	**50.9**	44.8	47.2	**52.3**
	GSM-H	26.1	**36.2**	27.1	**36.9**	**63.3**	63.1	55.1	**59.8**
	SVAMP	68.8	**81.2**	71.0	**81.1**	89.4	**89.6**	87.8	**89.0**
	ASDIV	73.1	**82.7**	74.3	**83.8**	91.0	**91.3**	89.0	**89.2**
	MAWPS	90.9	**92.2**	92.0	**93.7**	97.0	**97.1**	93.5	**94.9**
	STEM	49.7	**56.3**	57.0	**57.1**	68.0	**72.6**	67.2	**68.3**
	TABMWP	**57.9**	55.0	**63.6**	57.4	**73.0**	72.3	56.9	**66.2**
	SAT	56.2	**59.4**	59.4	**68.8**	80.0	**93.8**	81.2	**84.4**
	Average	54.8	**63.4**	57.4	**64.8**	77.6	**78.8**	72.9	**76.4**
Code	L.C.Bench-O	2.0	**30.5**	1.6	**11.8**	40.3	**49.8**	14.0	**44.8**
Logical	Zebra Puzzle	0.1	**7.4**	2.3	**7.4**	**2.9**	2.0	3.0	**4.0**
	KORBench	**21.8**	**21.8**	**21.9**	20.6	33.4	**35.4**	**32.0**	30.0
	Ruletaker	2.8	**42.4**	5.3	**45.6**	61.2	**63.5**	46.1	**60.2**
Algorithm	Graphwiz	4.5	**46.0**	1.9	**33.9**	36.6	**50.5**	38.5	**43.8**
	GraphInstruct	35.2	**73.5**	33.0	**69.2**	33.1	**33.5**	33.8	**53.7**

Model performances under two-stage fine-tuning

Table 3: Model performances under two-stage fine-tuning. The base models are trained solely o
TuluSFT datasets. L.C.Bench refers to LiveCodeBench. The best results are in purple.

Type	Datasets	Llama-3-8B		Llama-3.1-8B		Qwen-2.5-7B		Qwen-2.5-Coder	
		Base	Ours	Base	Ours	Base	Ours	Base	Ours
Math	GSM8K	70.7	**74.0**	73.8	**76.1**	76.9	**77.0**	74.3	**80.8**
	MinMath	28.8	**33.8**	30.2	**33.2**	47.2	**51.8**	**47.6**	48.2
	MATH	28.6	**30.6**	30.8	**34.2**	49.6	**51.5**	45.0	**50.8**
	GSM-H	35.0	**35.6**	**39.3**	35.3	55.0	**55.6**	55.4	**59.8**
	SVAMP	**83.0**	79.4	79.6	**83.4**	79.7	**83.6**	81.1	**85.3**
	ASDIV	81.4	**81.9**	82.3	**85.7**	**86.5**	86.3	86.9	**88.9**
	MAWPS	**93.9**	93.5	**94.3**	93.1	95.6	**95.9**	95.4	**96.6**
	STEM	42.8	**52.9**	50.2	**58.9**	70.1	**71.9**	**69.3**	66.2
	TABMWP	65.8	**67.1**	**69.6**	57.2	77.3	**78.9**	**80.5**	78.3
	MATHQA	37.7	**54.7**	44.7	**60.2**	**80.7**	80.2	75.3	**77.2**
	SAT	40.6	**75.0**	65.6	**68.8**	90.6	**96.9**	**87.5**	84.4
	Average	55.3	**61.7**	60.0	**62.4**	73.6	**75.4**	72.6	**74.2**
Code	CRUX	32.9	**42.1**	33.5	**49.9**	46.5	**49.4**	52.8	**56.4**
	L.C.Bench	9.8	**14.2**	7.6	**11.3**	**25.8**	23.5	27.2	**29.9**
Logical	Zebra Logic	7.7	**10.5**	**8.1**	9.4	**9.4**	8.8	9.2	**10.8**
	KORBench	**27.0**	27.2	**27.0**	24.7	40.4	**41.0**	34.6	**39.4**
	Mmlu-Redux	51.8	**56.3**	53.8	**54.3**	64.2	**69.7**	**66.5**	64.2
	Ruletaker	60.4	**66.2**	61.5	**67.0**	73.2	**74.4**	73.1	**73.9**
Algorithm	Graphwiz	**44.0**	39.6	**44.5**	39.2	29.0	**33.2**	33.9	**35.4**
	GraphInstruct	27.5	**36.0**	29.8	**55.9**	32.4	**35.8**	31.5	**40.9**
	CLRS	17.2	**24.5**	16.8	**25.8**	36.0	**43.6**	42.2	**49.4**

RQ1

How important is the Chain of Execution (CoE) format compared to traditional I/O or solution-style supervision?

Table 4: Comparison of TracePile with alternative supervision strategies under two-stage fine-tuning on Qwen-2.5-Coder 7B. LC-O and Z.L denote LiveCodeBench-Output and Zebra Logic.

Methods	**Stage-1 Data**	**GSM8K**	**MATH**	**LC-O**	**Z.L**
Baseline	–	74.3	45.0	29.8	9.2
Pure Code Solution	OpenMathIns. 1	80.1	46.1	19.4	5.6
Pure Math Solution	OpenMathIns. 2	**82.1**	**51.1**	17.4	6.6
Generic Instruction	WebInstruct	81.3	49.0	18.8	9.1
CodeI/O	CodeI/O	79.3	39.8	17.6	8.6
Ours	TracePile	80.8	50.8	**35.5**	**10.8**

RQ2

What components contribute most to TracePile's effectiveness?

Table 5: Ablation studies under two-stage fine-tuning. Average performances of each category are reported.

Methods	Mathematical	Code	Logical	Algorithm
Ours	**62.4**	**30.6**	**38.9**	40.3
w/o mathematical	60.6	26.2	38.4	**40.6**
w/o algorithm	61.5	23.1	38.0	33.2
w/o competition	62.0	24.9	37.8	35.0
w/o diversification	61.8	27.6	38.0	37.2
w/o query	62.0	30.0	38.7	37.9
w/o code	62.2	28.2	38.3	39.5

RQ3

Does performance scale with more TracePile data?

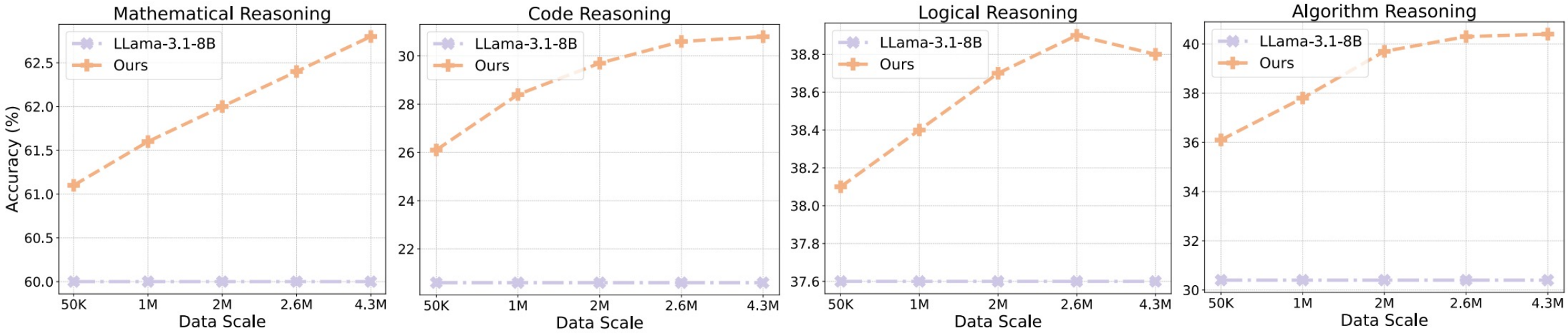

Figure 3: Model performances with scaling the size of CoE samples.

RQ4

What types of reasoning are improved by TracePile? Does it enhance robustness, reduce typical reasoning errors, and generalize?

Table 6: Compare ours with Qwen-2.5-base under two-stage fine-tuning in BBH subsets.

Modelsm	Tracking Shuffled Objects	Arithmetic	Logical Deduction	Web of Lies
Base	68.4	72.0	78.4	87.2
Ours	**87.6**	**82.0**	**83.6**	**92.8**

Takeaways

- CoE supervision bridges code and reasoning.
- TracePile provides large-scale procedural supervision.
- Enables transferable reasoning across math, code, and logic.
- A step toward general reasoning in LLMs.