7. Bridging between Code and Natural Language

Bridging Between Code and Natural Language

- **Code as mental scaffold for Reasoning**
  - Programming structures that shape and support human reasoning processes
- **Code as interactive tool for Reasoning**
  - Direct use of executable code to think through and solve problems
- **Code as data foundation for Reasoning**
  - The foundational role of code data in building capabilities

Scratchpad: Step-by-Step in Code

➢ When prompted to perform operations *"step by step"* and display intermediate steps in a "scratchpad", language models can successfully handle complex computations

DIRECT EXECUTION PREDICTION

```
Consider the following Python function:

def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s

Fill in the ??? below:
assert remove_Occ("PHP","P") == ???
```

Direct **Reasoning** in Code

SCRATCHPAD TRACING

```
Consider the following Python function:

def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s

output = remove_Occ("PHP","P")

What is the execution trace?
```

Large Language Model

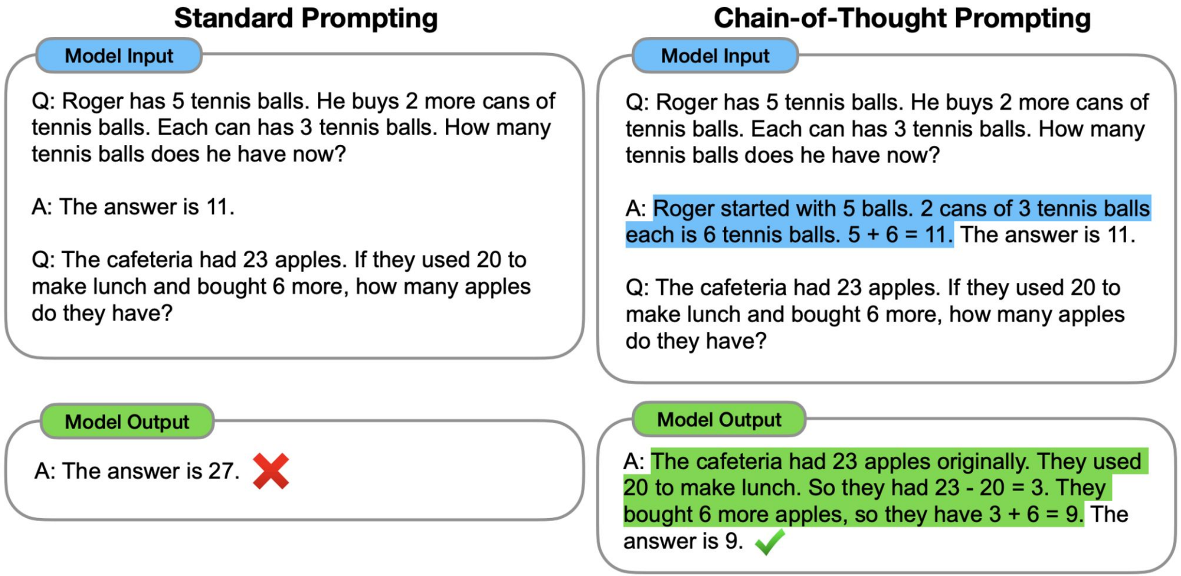
```
state: {}
line: def remove_Occ(s,ch):
state: {"remove_Occ": "<callable_object remove_Occ>"}
line: output = remove_Occ("PHP","P")
state: {"ch": "P", "s": "PHP"}
line:     for i in range(len(s)):
state: {"ch": "P", "s": "PHP", "i": 0}
line:         if (s[i] == ch):
state: {"ch": "P", "s": "PHP", "i": 0}
line:             s = s[0 : i] + s[i + 1:]
state: {"ch": "P", "s": "HP", "i": 0}
line:             break
state: {"ch": "P", "s": "HP", "i": 0}
line:     for i in range(len(s) - 1,-1,-1):
state: {"ch": "P", "s": "HP", "i": 1}
line:         if (s[i] == ch):
state: {"ch": "P", "s": "HP", "i": 1}
line:             s = s[0 : i] + s[i + 1:]
state: {"ch": "P", "s": "H", "i": 1}
line:             break
state: {"ch": "P", "s": "H", "i": 1}
line:     return s
state: {"remove_Occ": "<callable_object remove_Occ>",
 "output": "H"}
```

Step-by-Step **Reasoning** in Code

Nye et al., "Scratchpads for Intermediate Computation with Language Models", 2021

Chain-of-Thought: Step-by-Step in NL

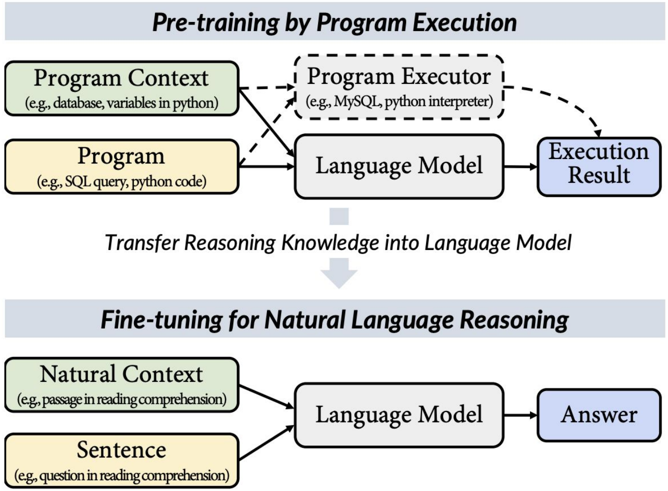
- ➢ Chain-of-thought prompting incorporates reasoning steps in the prompt and significantly improves language model performance on reasoning tasks

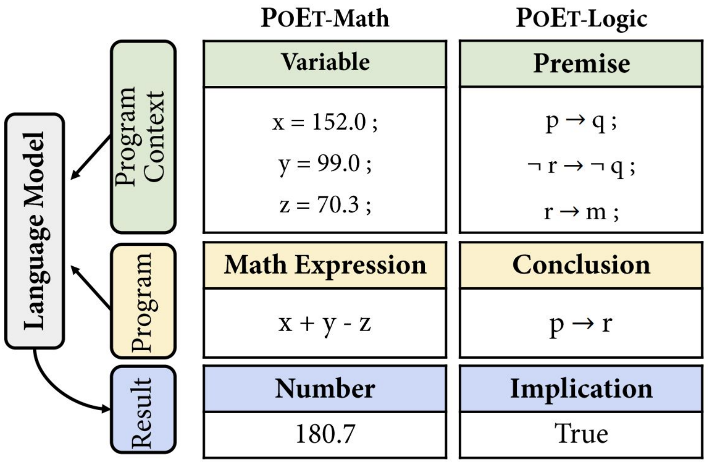


Wei et al., Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, 2022

POET: Reasoning Like Program Executors

➢ POET teaches language models to improve natural language reasoning by learning from programs and their execution results.

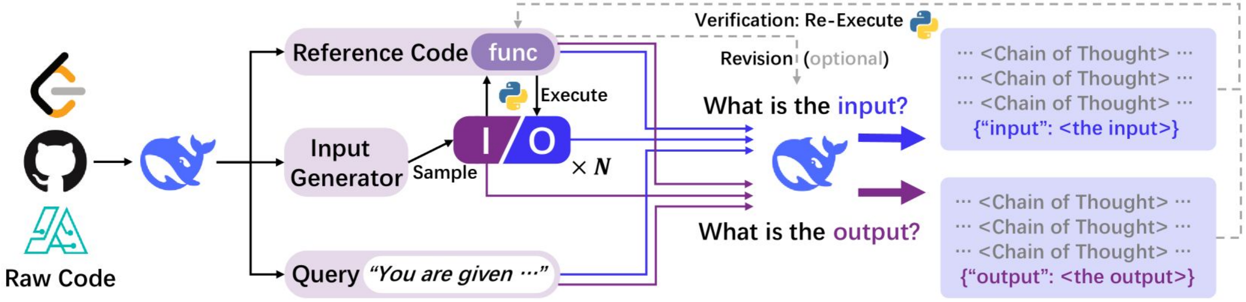




Pi et al., "Reasoning Like Program Executors", EMNLP 2022

Code I/O: Reasoning via Code Input-Output Prediction

➢ Code I/O improves language models' reasoning abilities across diverse tasks by having them predict code inputs and outputs in natural language.

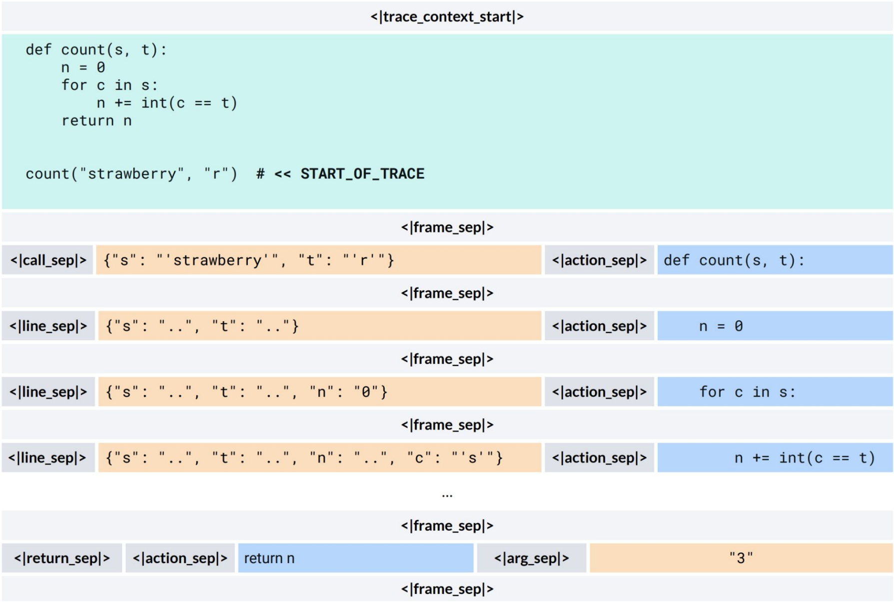


1st Stage Dataset	# (M)	Wino Grande	DROP	GSM 8K	MATH	GPQA	MMLU -STEM	LC -O	CRUX -I	CRUX -O	BBH -EN	BBH -ZH	Zebra Logic	Kor Bench	Live Bench	AVG
Qwen 2.5 Coder 7B																
2nd Stage Only		66.9	70.7	83.4	71.6	41.5	77.2	20.7	61.3	60.0	68.3	70.6	10.9	38.7	26.0	54.8
CODEI/O	3.5	67.9	76.4	86.4	71.9	43.3	77.3	23.7	63.6	64.9	69.3	72.8	10.7	44.3	28.5	57.2
CODEI/O++	3.5	66.9	79.1	85.7	72.1	40.6	77.9	24.2	62.5	67.9	71.0	74.2	10.7	45.7	29.1	57.7

Li et al., "CodeI/O: Condensing Reasoning Patterns via Code Input-Output Prediction", 2025, arXiv preprint arXiv:2502.07316v2.

CWM: Code World Model

➢ Given a source code context and a marker of the trace starting point, CWM predicts a series of stack frames representing the Program states and the actions (executed code).



FAIR CodeGen Team et al. (2025). CWM: An Open-Weights LLM for Research on Code Generation with World Models. arXiv preprint arXiv:2510.02387.

PAL: Program Aided Reasoning

- ➢ PAL uses LLMs to decompose NL problems into programmatic steps, then offloads execution to a Python interpreter to avoid arithmetic errors.

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Chain-of-Thought

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls.
```
tennis_balls = 5
```
2 cans of 3 tennis balls each is
```
bought_balls = 2 * 3
```
tennis balls. The answer is
```
answer = tennis_balls + bought_balls
```

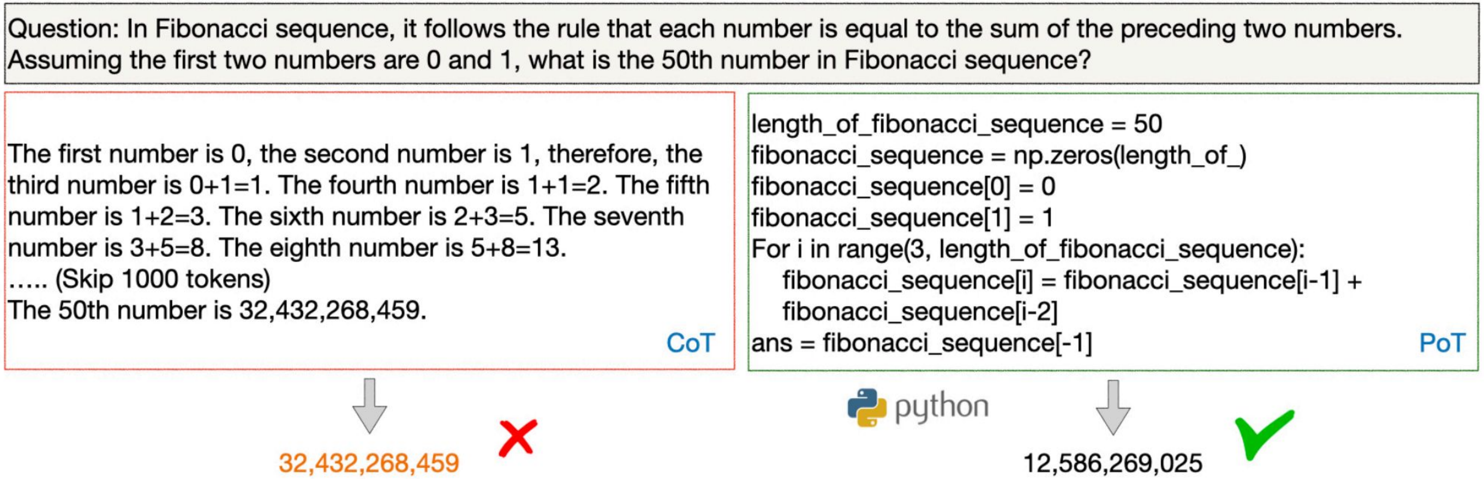
Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Program-Aided Language Models

Gao et al. "PAL: Program-Aided Language Models." arXiv preprint arXiv:2211.10435

PoT: Program of Thoughts

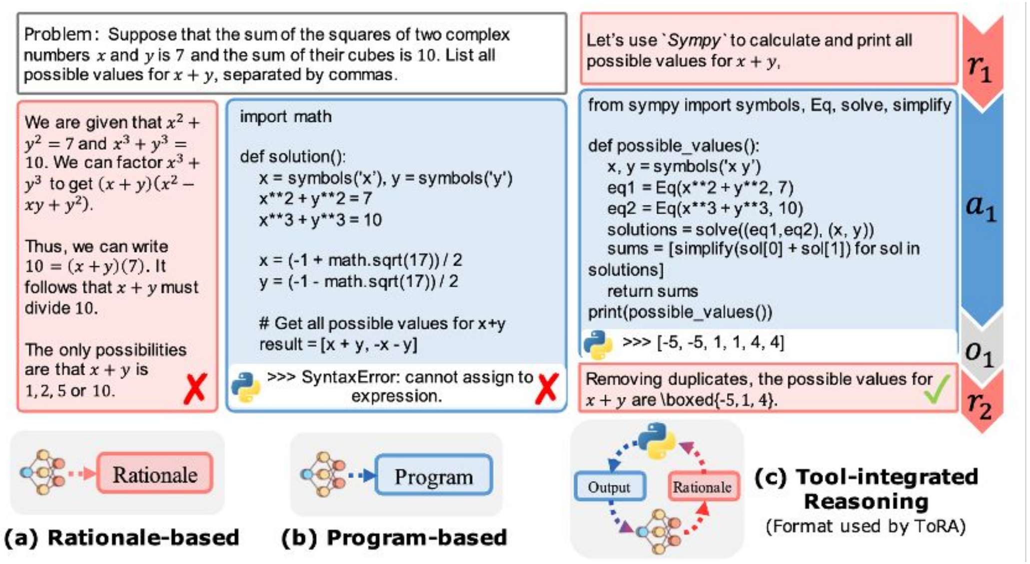
- PoT disentangles computation from reasoning by using LLMs to express reasoning as executable programs, while delegating all calculations to an external computer



Chen et al. *"Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks." Transactions on Machine Learning Research, 2023*

ToRA: Tool-Integrated Reasoning

➢ ToRA interleaves natural language reasoning with program-based tool calls, combining the strengths of semantic analysis and precise computation



Gou et al. "ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving." International Conference on Learning Representations (ICLR), 2024

SimpleTIR: RL Enables Multi-Turn TIR

- SimpleTIR stabilizes multi-turn tool-integrated reasoning by filtering out “void turns” that generate neither code nor answers, and train LMs via end-to-end RL.

Xue et al. “SimpleTIR: End-to-End Reinforcement Learning for Multi-Turn Tool-Integrated Reasoning.” arXiv preprint arXiv:2509.02479, 2025

The Code & Language Mixture for Pre-training

- While balancing **Code**, **Math**, and **Text** data is crucial for pre-training, limited evidence exists on how this balance scales to large datasets.

- Experimental results from Qwen2.5-Coder indicate that 7:2:1 (Code:Text:Math) achieves a good balance.

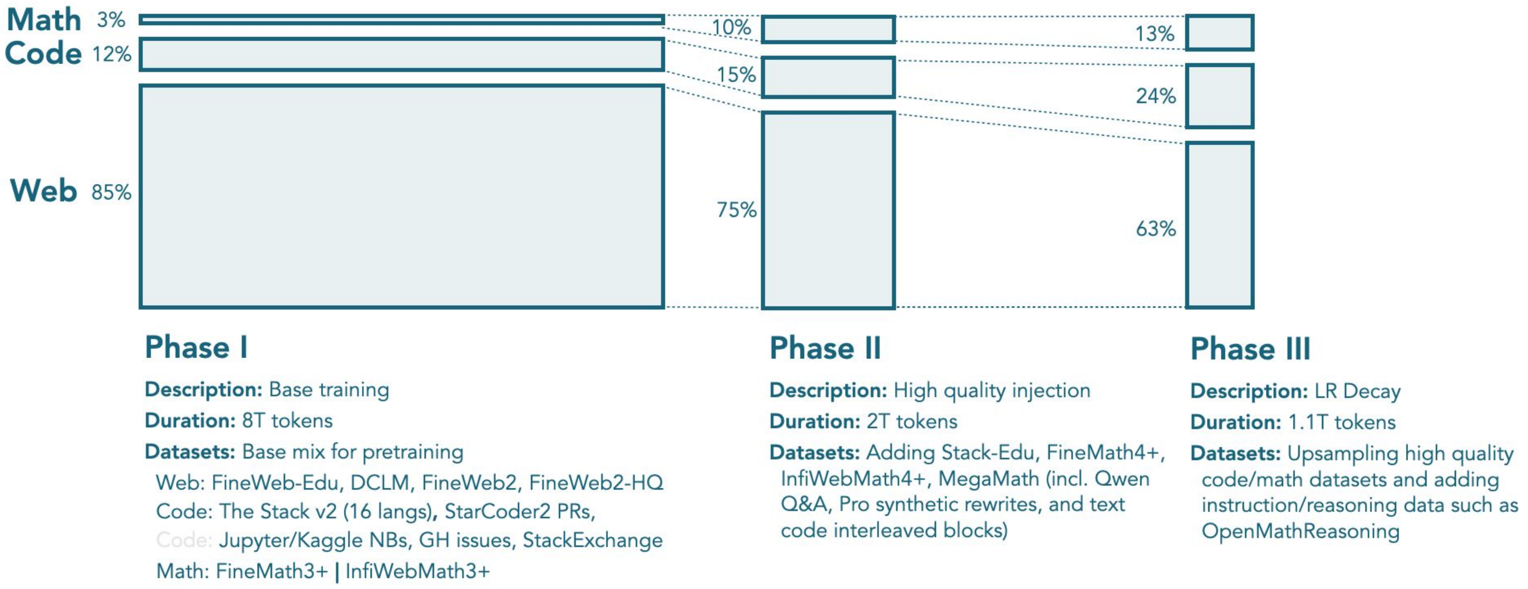
Token Ratio			**Coding**		**Math**		**General**			*Average*
Code	Text	Math	Common	BCB	MATH	GSM8K	MMLU	CEval	HellaSwag	
100	0	0	**49.8**	**40.3**	10.3	23.8	42.8	35.9	58.3	31.3
85	15	5	43.3	36.2	26.1	52.5	56.8	57.1	70.0	48.9
70	20	10	48.3	38.3	**33.2**	**64.5**	**62.9**	**64.0**	**73.5**	**55.0**

Table 3: The performance of Qwen2.5-Coder training on different data mixture policy.

Hui et al., 2024. Qwen2.5-Coder Technical Report. arXiv:2409.12186.

The Code & Language Mixture for Pre-training

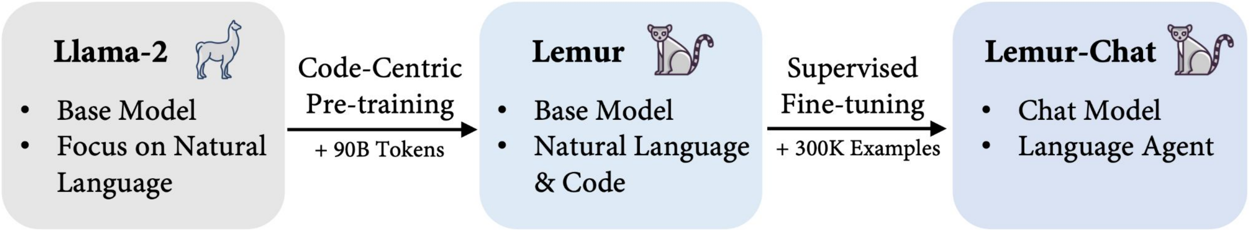
- ➢ Three-stage pretraining (11.1T tokens total) gradually upsampled **Math** and **Code** data while reducing web content, then applied mid-training for specialized capabilities.



Elie et al., 2025. "SmolLM3: smol, multilingual, long-context reasoner". https://huggingface.co/blog/smollm3

The Code & Language Mixture for CPT

➢ Lemur paper found that a 10:1 (Code:Text) ratio works well for Llama's continual pre-training (CPT), but predicting optimal data mixture ratios remains challegning



Model	Text			Code					Avg
	QA	Reason	Math	Python		SQL	MCode	DS	
	MMLU	BBH	GSM8K	HE	MBPP	Spider	MultiPL-E	DS-1000	
StarCoder-15B	30.8	33.2	8.9	33.6	52.7	58.3	25.3	26.0	33.6
StarCoderPlus-15B	42.0	36.2	17.7	26.2	37.0	48.8	21.4	19.4	31.1
CodeLlama-34B	52.8	42.2	32.7	48.8	55.0	68.4	36.4	31.8	46.0
Llama-2-70B	68.9	51.2	56.8	30.5	45.4	60.0	24.4	11.3	43.6
Lemur-70B	64.5	51.6	54.9	35.4	53.2	62.8	30.4	30.7	**47.9**

Xu et al., 2024. "Lemur: Harmonizing Natural Language and Code for Language Agents". arXiv:2310.06830.

Influence from Code to Reasoning

➢ Reasoning depends more on patterns of procedural demonstration than on memorised answers. For reasoning, key sources consist of maths, StackExchange, ArXiv, and **code**.

Positively influential code

```
function eqOfLine(x1, y1, x2, y2) {
  if (x1 === x2) {
    // Handle a vertical line
    return `x = ${x1}`;
  } else {
    // Calculate the slope
    const m = (y2 - y1) / (x2 - x1);
    const b = y1 - m * x1;
    // Return y = mx + b
    return `y = ${m}x + ${b}`;
  }
}
```

Positively influential math

If a straight line passing through the points $P(x_1, y_1), Q(x_2, y_2)$ is making an angle θ with the positive X-axis, then the slope of the straight line is:

(A) $\frac{y_2+y_1}{x_2+x_1}$
(B) θ
(C) $\frac{y_2-y_1}{x_2-x_1}$
(D) $\sin\theta$

Solution:
Correct answer: (C)

Ruis et al., 2024. "Procedural Knowledge in Pretraining Drives Reasoning in Large Language Models". arXiv:2411.12580v2.