PENCIL: Long Thoughts with Short Memory

Chenxiao Yang

@FLaNN

May 26th, 2025

Nathan Srebro

David McAllester

Zhiyuan Li

Chain-of-Thought (CoT)

CommonsenseQA

Q: Sammy wanted to go to where the people were. Where might he go?
Options: (a) race track (b) populated areas (c) desert (d) apartment (e) roadblock

A: The answer must be a place with a lot of people. Race tracks, desert, apartments, and roadblocks don't have a lot of people, but populated areas do. So the answer is (b).

StrategyQA

Q: Yes or no: Would a pear sink in water?

A: The density of a pear is about 0.6 g/cm^3, which is less than water. Thus, a pear would float. So the answer is no.

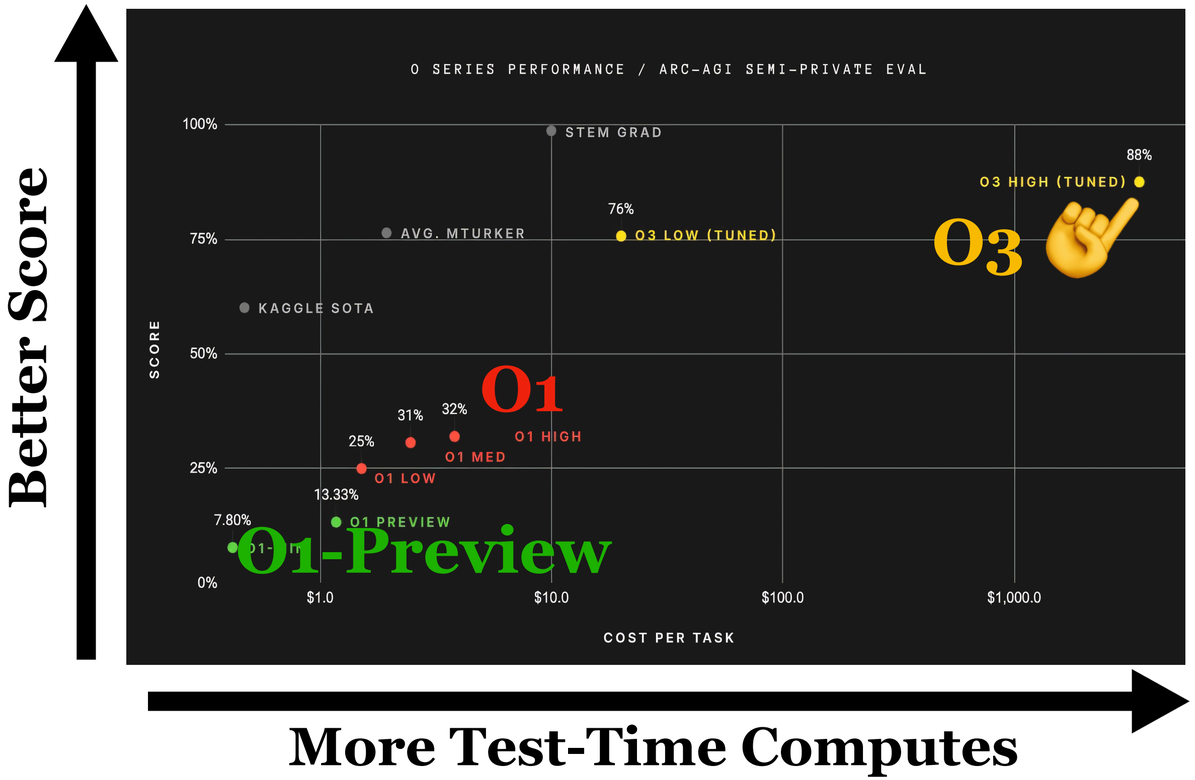
Date Understanding

Q: The concert was scheduled to be on 06/01/1943, but was delayed by one day to today. What is the date 10 days ago in MM/DD/YYYY?

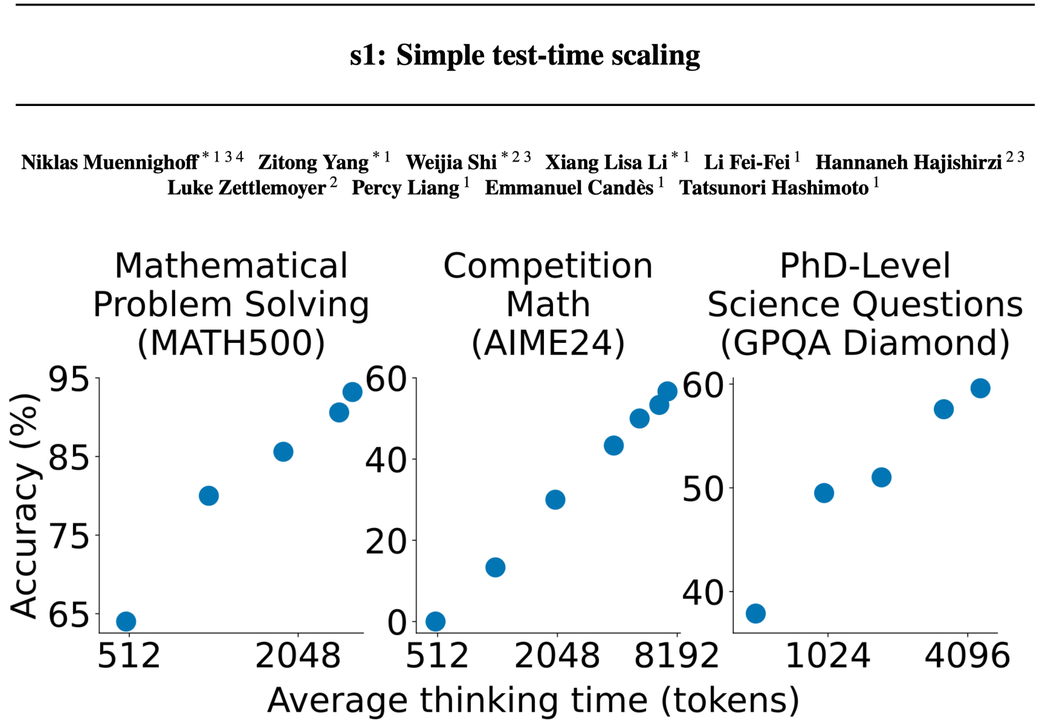
A: One day after 06/01/1943 is 06/02/1943, so today is 06/02/1943. 10 days before today is 05/23/1943. So the answer is 05/23/1943.

Chain-of-Thought generates a series of **thoughts** before providing the final answer.

The Power of Long CoT



Performance of O3 on ARC-AGI-Pub



S1, Muennighoff et al., 2025

Long CoT empowers transformers to solve computationally more intensive tasks (Feng et al. 2024, Merill et al. 2024, Li et al. 2024, etc.)

Long CoT is great! BUT …

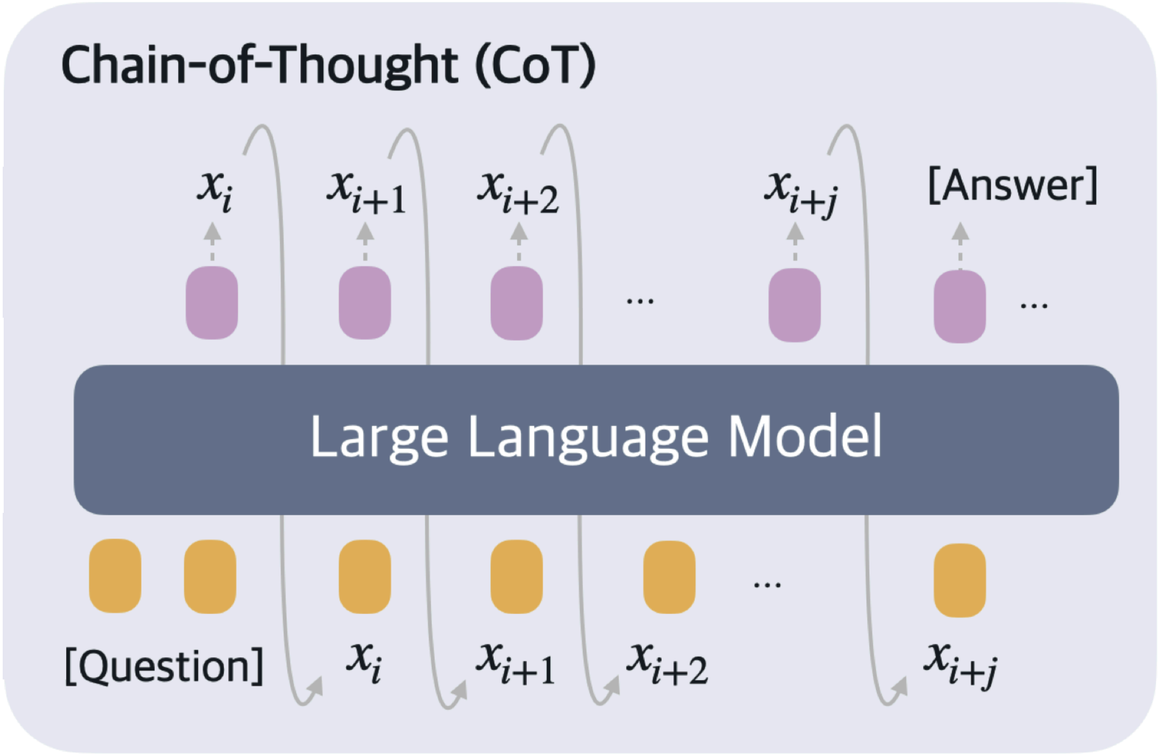


Image adapted from Hao et al., 2024

- **Out-of-Memory:** Accumulated thoughts will exceed model's effective context windows.
- **Large Cost:** Self-attention makes generating each new token prohibitively expensive.
- **Low Accuracy:** Relevant information becomes buried and hard to retrieve (Liu et al. 2024, Ye at al. 2025, etc.)

Most Intermediate Steps are "Erasable"

- **Math**

Theorem.
There are infinitely many prime numbers.

Lemma.
Every integer n> 1 has a prime p divisor.

Proof of Lemma. Take the smallest divisor p>1 of n. If p were composite, it would factor as ab with 1<a,b<p, contradicting minimality. Hence p is prime and divides n. ∎

No longer useful

Proof of Theorem. Assume, for contradiction, that the complete list of primes is p_1, ... ,p_k. Set N = p_1p_2 ... p_k + 1. **By the lemma**, N has some prime divisor q. But every p_i leaves remainder 1 when dividing N, so none of them divides N; hence q is a new prime, contradicting completeness. Therefore primes are infinite. ∎

- **LLM Reasoning** **(from DeepSeek-R1 run)**

Question: "Your mother has 4 children: North, South, East—what's the 4th child's name?"

Alright, since the three children already named —North, South, and East—correspond precisely to three of the four cardinal directions, it seems perfectly logical to complete the compass pattern by concluding that the fourth child must therefore be named **West**.

No longer useful

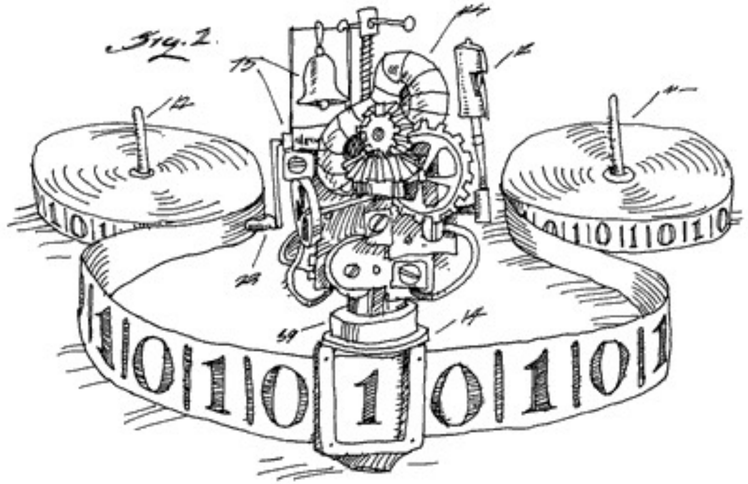
Wait — maybe it's a trick: "your mother" means *you* are the fourth child.

Wait — the question wants a *name*, so the answer is actually *your own name*.

→ **Final answer**: your name.

“Erasure” is Fundamental to Computation

One Step of Thinking ⇔ One Step of Computation



Turing Machine (1936)

Modern Computer

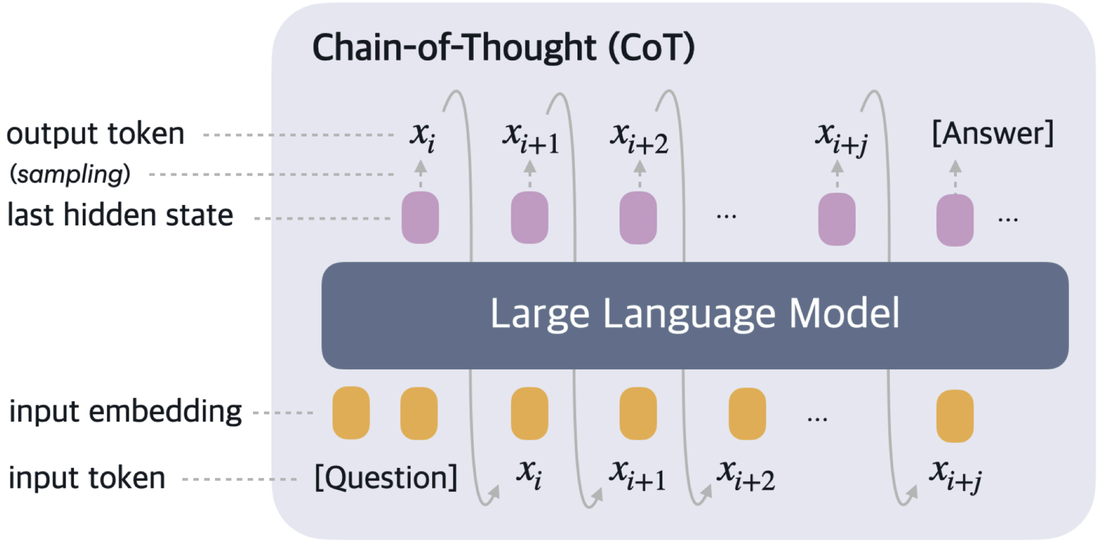
Transformer + **PENCIL**

PENCIL

PENCIL ENables Context-efficient Inference and Learning

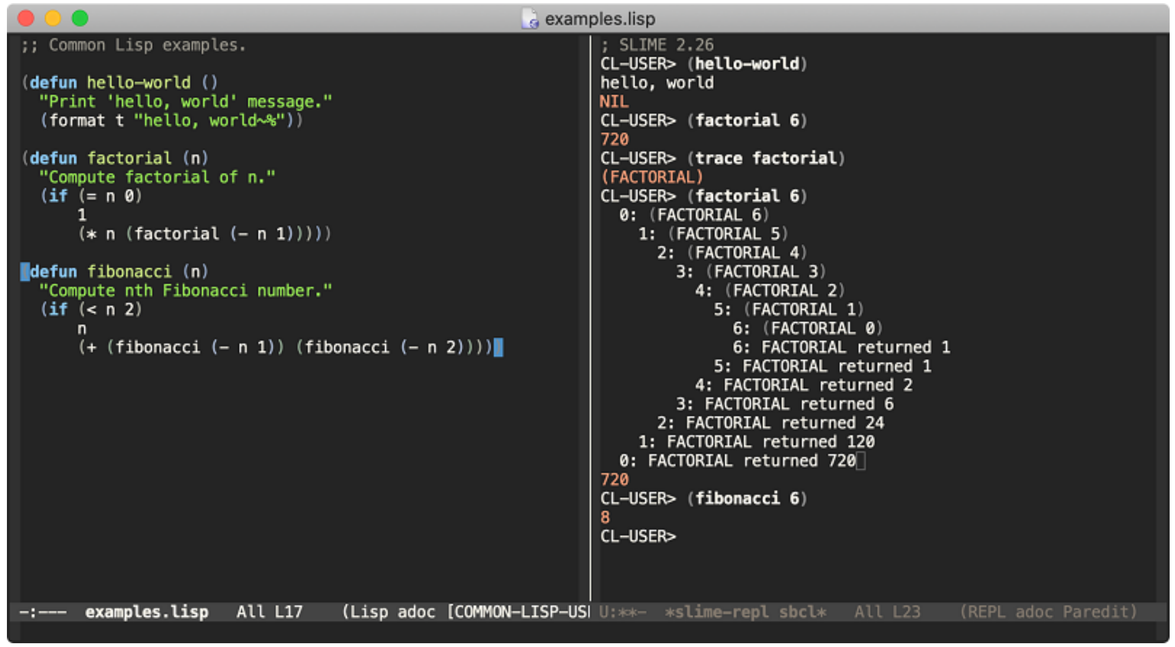
What is ✏️ PENCIL?

Model Generation (Write)



Autoregressive Next-token Generation

Reduction Rule (Erase)



Functional Programming

What is ✏️ PENCIL?

Model Generation (Write)

Reduction Rule (Erase)

C [CALL] **T** [SEP] **A** [RETURN] ⇒ **C A**

Context ∈ Σ*

Thoughts ∈ Σ*

Answer ∈ Σ*

Special Tokens

- Reduction is triggered when the sequence matches the pattern
- PENCIL iteratively  generate thoughts and triggers reduction.

Example 1: Arithmetic Expression Evaluation

A toy store put together party bags for a birthday. They made 3 blue bags with 5 toys each and 2 red bags with 4 toys each. How many toys were used in total? **There were 23 toys used in total**.own into parts! **There were 15 toys in all blue bags.** [CALL] [SEP] [RETURN] [SEP] **There were 15 toys in all blue bags** [SEP] There were 23 toys used in total. [RETURN] There were 8 toys in all red bags. [RETURN]

C [CALL] **T** [SEP] **A** [RETURN] ⇒ **C A**

Example 1: Arithmetic Expression Evaluation

Prompt: A toy store put together party bags for a birthday. They made 3 blue bags with 5 toys each and 2 red bags with 4 toys each. How many toys were used in total?

PENCIL 

A toy store put together party bags for a birthday. They made 3 blue bags with 5 toys each and 2 red bags with 4 toys each. How many toys were used in total?

Example 1: Arithmetic Expression Evaluation

Prompt : A toy store put together party bags for a birthday. They made 3 blue bags with 5 toys each and 2 red bags with 4 toys each. How many toys were used in total?

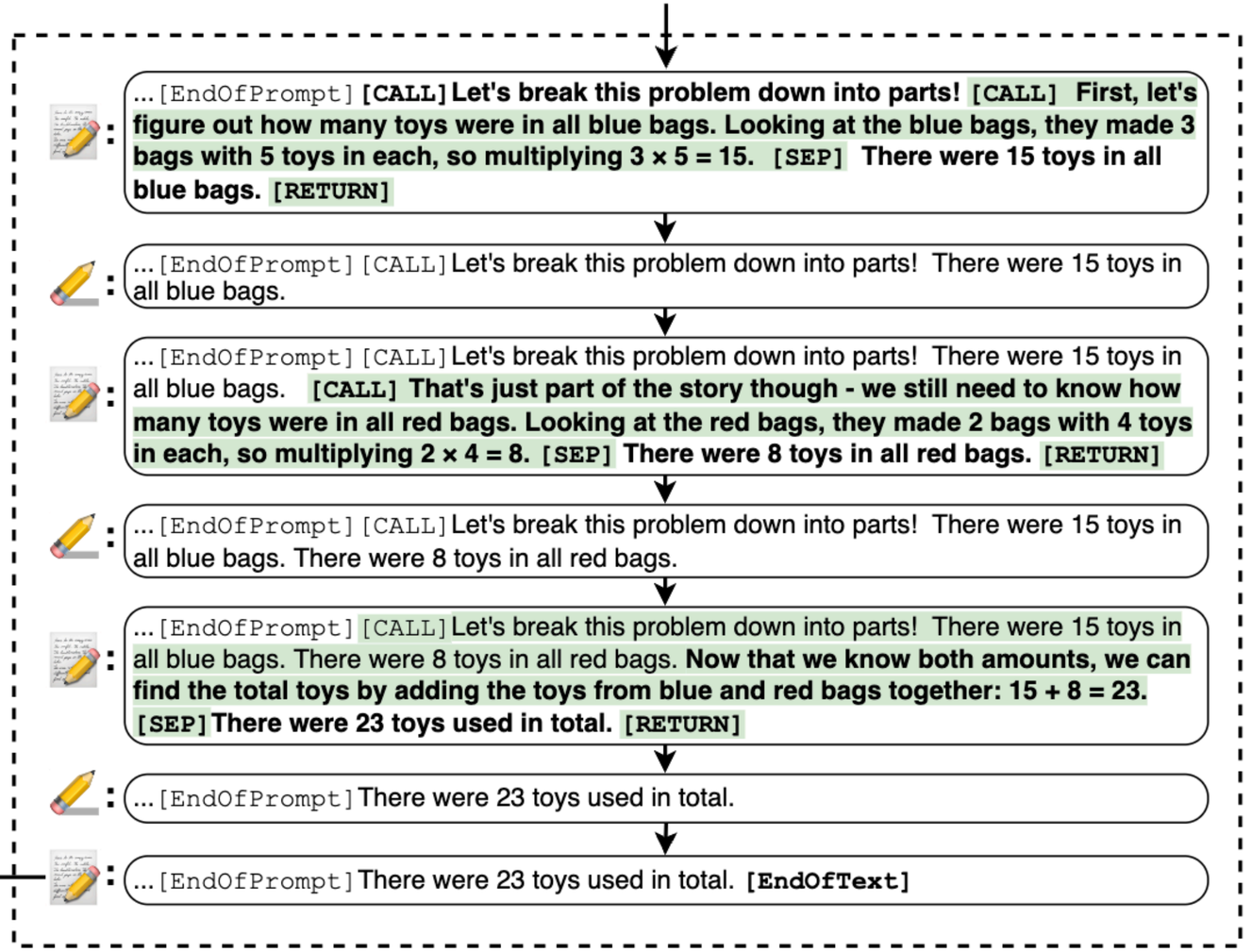
Chain-of-Thought

Response : Let's break this problem down into parts! First, let's figure out how many toys were in all blue bags. Looking at the blue bags, they made 3 bags with 5 toys in each, so multiplying 3 × 5 = 15. There were 15 toys in all blue bags. That's just part of the story though - we still need to know how many toys were in all red bags. Looking at the red bags, they made 2 bags with 4 toys in each, so multiplying 2 × 4 = 8. There were 8 toys in all red bags. Now that we know both amounts, we can find the total toys by adding the toys from blue and red bags together: 15 + 8 = 23. There were 23 toys used in total.

PENCIL

Response : There were 23 toys used in total.

Hidden CoT



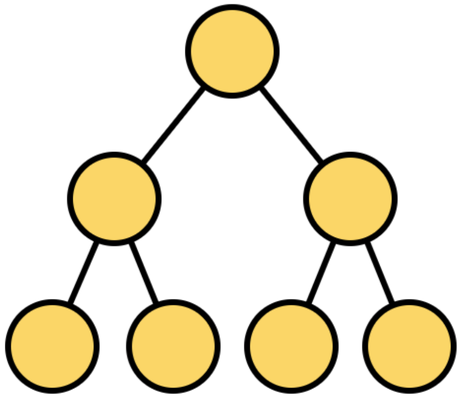
Example 2: Quantified Boolean Formula (QBF)

$\exists X_2 \forall X_1 : (X_2 \vee \neg X_2 \vee X_1) \wedge (X_1 \vee X_2) \wedge (X_2) \wedge (\neg X_2 \vee \neg X_1) \wedge (X_1 \vee \neg X_1) \wedge (\neg X_1 \vee \neg X_2)$ Answer is True

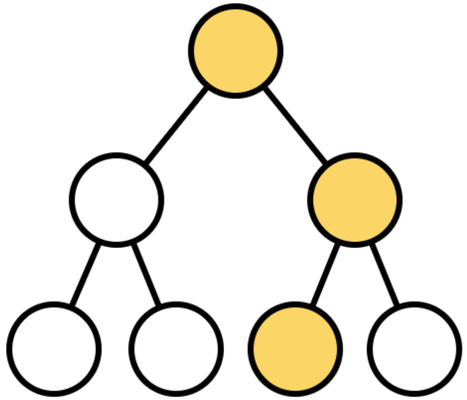
[CALL] Try X_2=False [CALL] Try X_1=False, 1st clause is True, 2nd clause is False [RETURN] [CALL] Try X_1=True, 3rd clause is True, 4th clause is False [RETURN] [RETURN]

[CALL] Try X_2=True [CALL] Try X_1=False, 1st clause is True, 2nd clause is True [SEP] [CALL] Try X_1=True, 3rd clause is True, 4th clause is True [SEP] Answer is True [RETURN] [RETURN]

[SEP] Answer is True [RETURN]

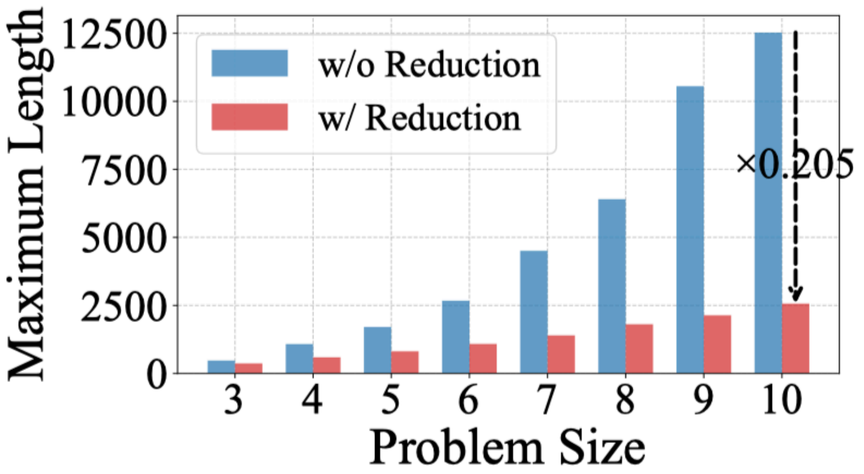


🪶 **CoT**: length $\propto \exp(n)$

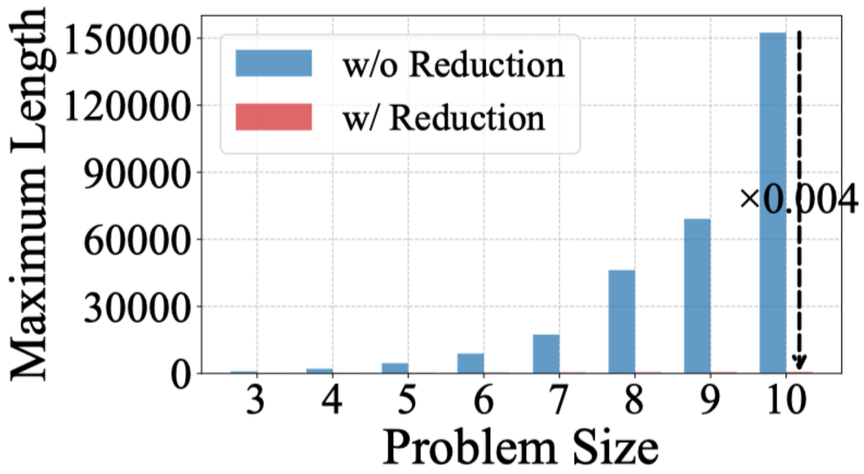


✏️ **PENCIL**: length $\propto \text{poly}(n)$

Max Sequence Length Comparison (CoT v.s. PENCIL)



3-SAT
(a special case of QBF)



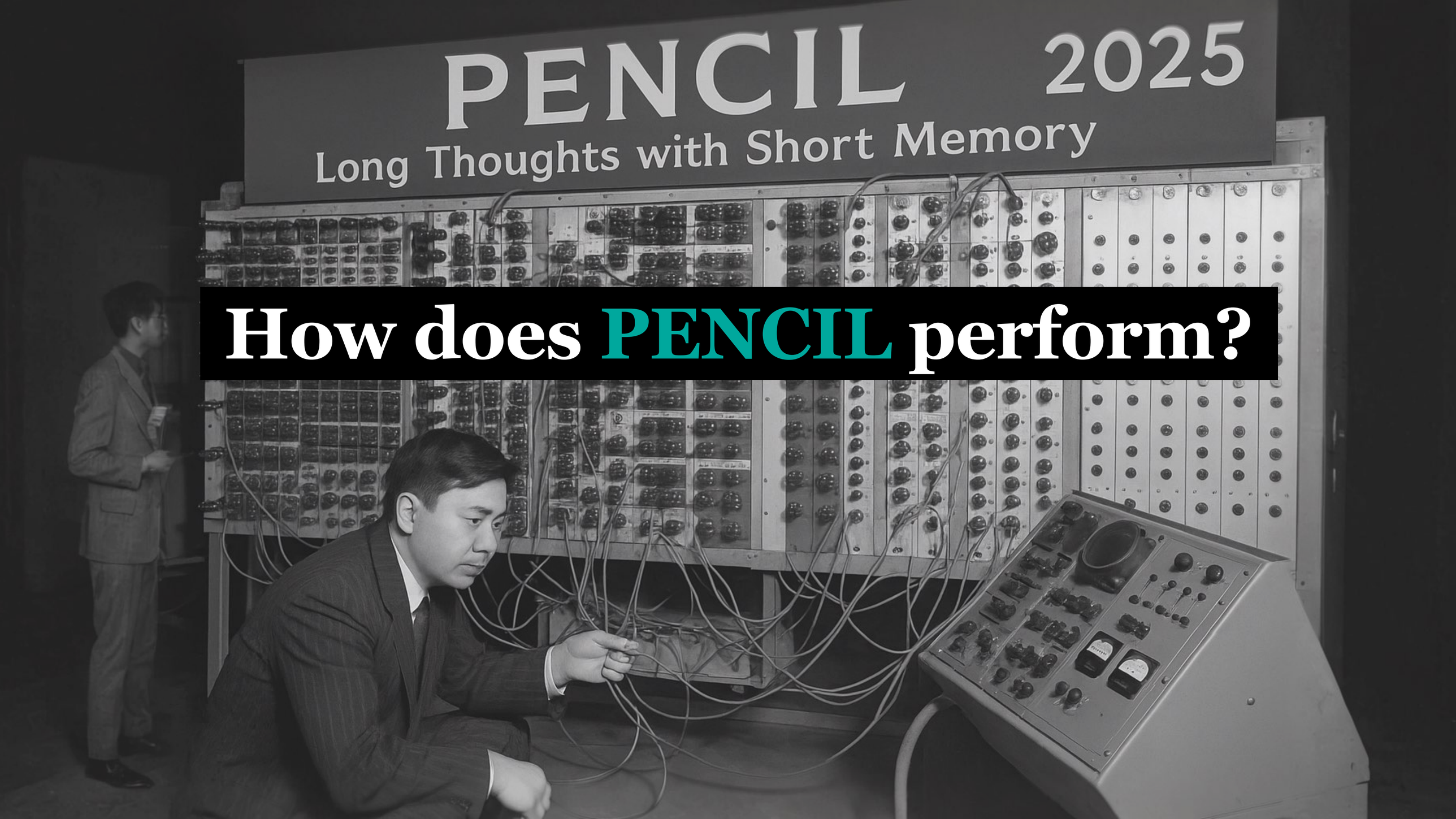
QBF

PENCIL significantly reduces the maximal context length during inference

(× 0.004 when n=10 on QBF)

Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought

Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought
Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought Thought



How does PENCIL perform?

Experimental Setting: Training and Inference

Training: Key difference between CoT and PENCIL ⇒ **Data Generation***

$$\mathscr{L}_{\textbf{CoT}} = -\sum \log p(\textbf{next token} \mid \textbf{complete sequence})$$

$$\mathscr{L}_{\textbf{PENCIL}} = -\sum \log p(\textbf{next token} \mid \textbf{reduced sequence})$$

We train a small transformer (25M parameter, 2048 context length) from scratch.

Inference: **C** [CALL] **T** [SEP] **A** [RETURN] ⇒ **C A**

Preserve the KV cache of **Context (C)** and recompute that for **Answer (A)**

** Datasets are generated by running specialized algorithms*

Performance Comparison on 3-SAT and QBF

n =	3	4	5	6	7	8	9	10
Baseline	66	57	46	51	46	51	49	51
CoT	100	100	100	99	84	63	54	50
PENCIL	100	100	100	99	99	100	100	100

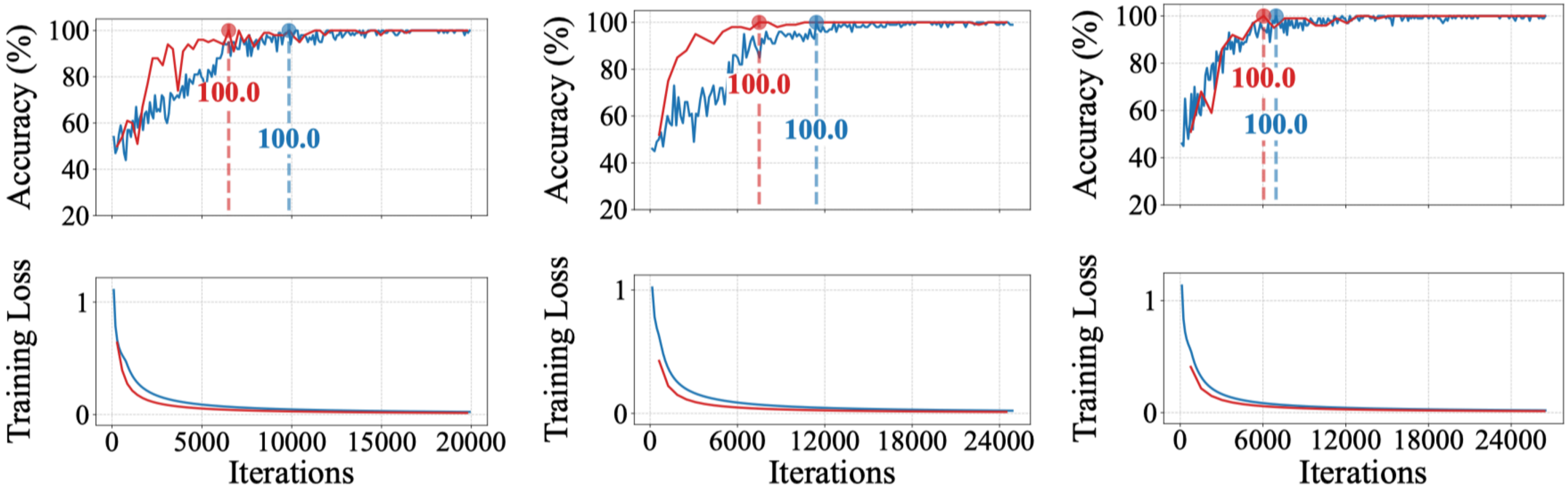
3-SAT

n =	3	4	5	6	7	8	9	10
Baseline	90	82	85	68	60	69	71	66
CoT	100	100	97	94	74	72	69	73
PENCIL	100	100	100	100	100	100	100	100

QBF

✏️**PENCIL** significantly outperforms 🖋️**CoT** on NP-hard tasks SAT and QBF (i.e. almost perfect v.s. random guessing).

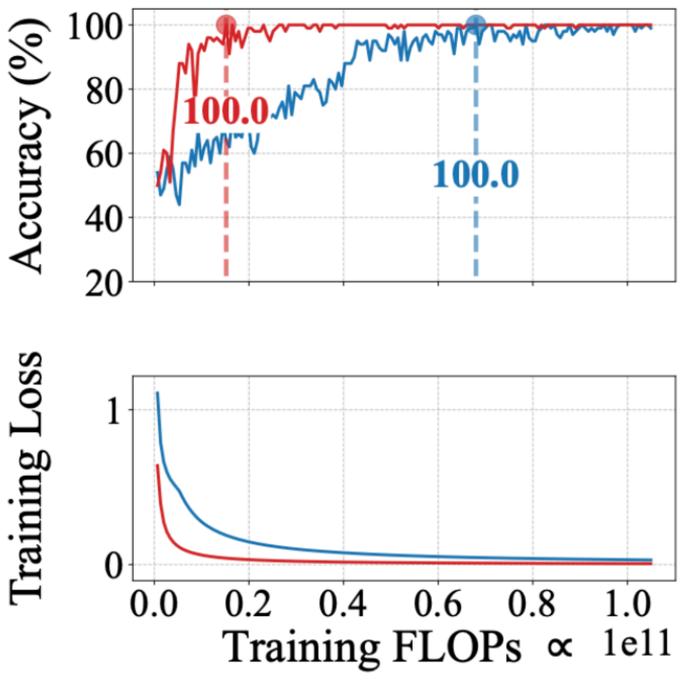
Convergence Speed (CoT v.s. PENCIL)



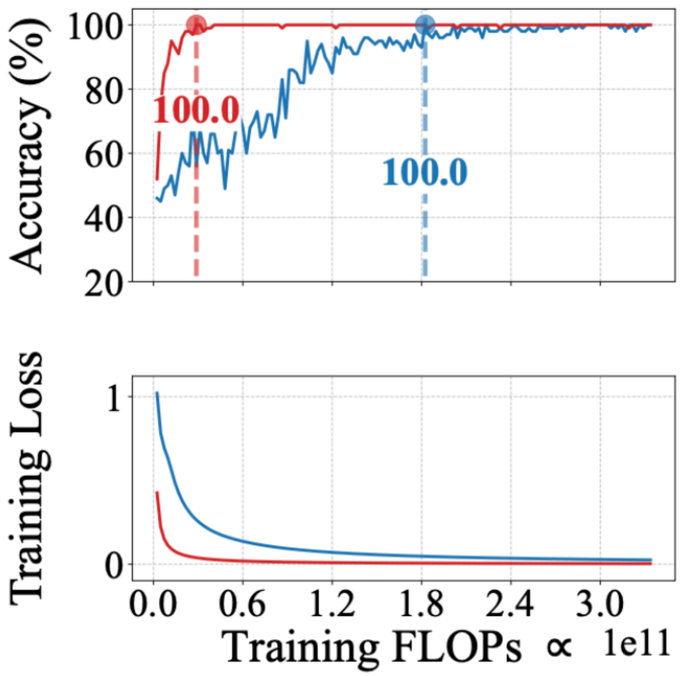
QBF n=4 ***QBF n=5*** ***QBF n=6***

✏️**PENCIL** converges faster and is more sample efficient.

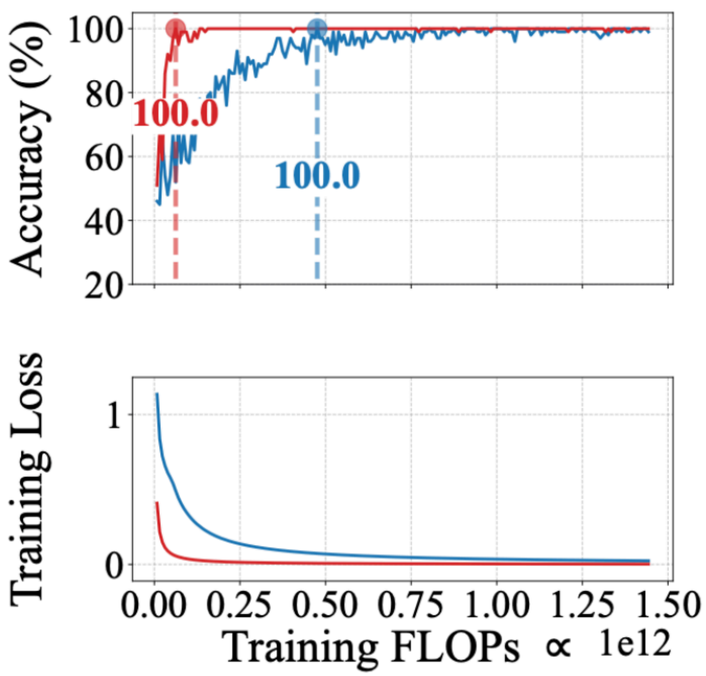
For Each Token, len(prefix for ✏️) << len(prefix for 🪶)



QBF n=4



QBF n=5



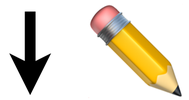
QBF n=6

✏️**PENCIL** is computationally more efficient than 🪶**CoT**.

Example 3: Einstein's Puzzle

- **Constraint 1 :** The green house is immediately to the right of the one who keeps birds
- **Constraint 2 :** The Brit is immediately to the right of the German
- **Constraint 3 :** The one who keeps dogs is the same house as the red house
- **Constraint 4 :** The one who keeps birds is immediately to the right of the Swede

Question: who owns the fish?



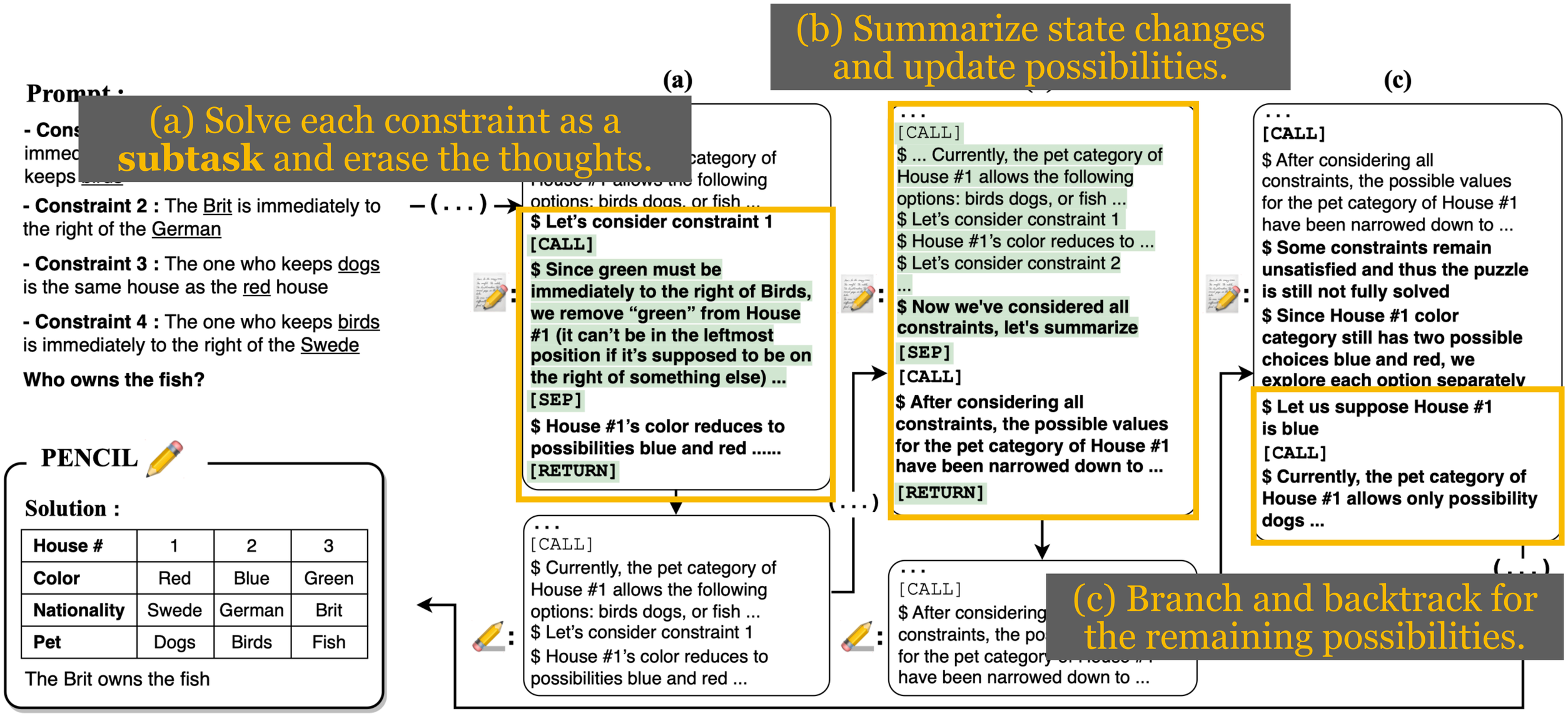
Solution :

House #	1	2	3
Color	Red	Blue	Green
Nationality	Swede	German	Brit
Pet	Dogs	Birds	Fish

Answer: the Brit owns the fish

** The original Einstein's Puzzle has 5 categories*

Example 3: Einstein's Puzzle



Special Usage: Summarization

C [CALL] **T** [SEP] [CALL] **T'** [RETURN] ⇒ **C** [CALL] **T'**

Long Thoughts Summarized Thoughts

Tail recursion in functional programming:

python Copy

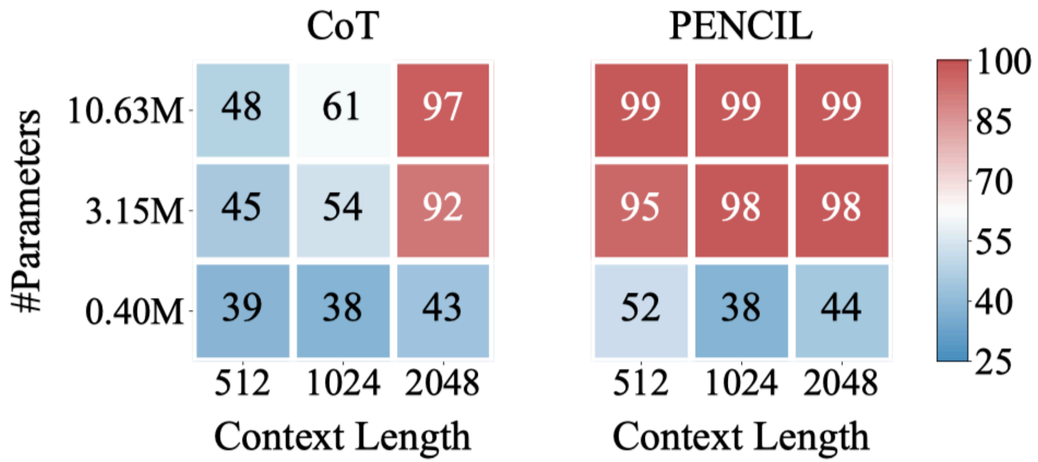
```python
def factorial_tail(n, acc=1):
    """Tail-recursive factorial: the recursive call is the final action."""
    if n == 0:
        return acc                  # base case
    return factorial_tail(n-1, acc*n)  # tail call

print(factorial_tail(5))  # → 120
```

The “returned value” is another “function call” ⇒ **A (Answer) = [CALL] T’**

Performance on Einstein's Puzzle

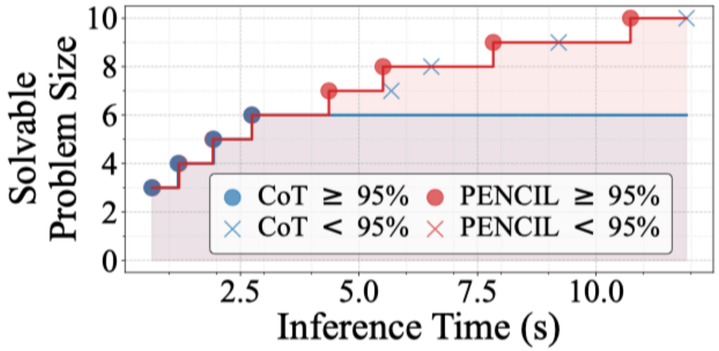
Puzzle Size	CoT	PENCIL
5 × 5	25	97
4 × 4	34	100
3 × 3	99	99



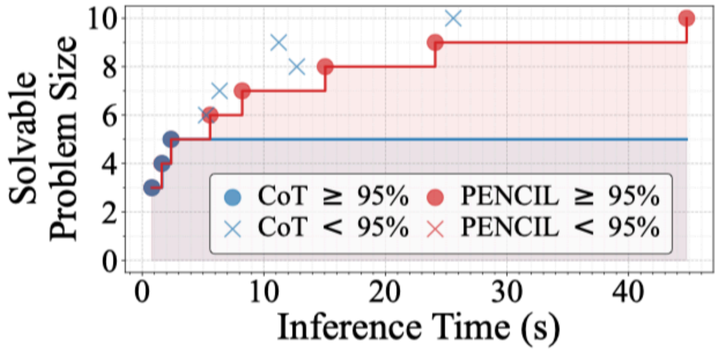
✏️**PENCIL** solves Einstein's puzzle almost perfectly – a logic puzzle that even GPT-4 struggles with.

(Max context length, **CoT** = 151,192 **PENCIL** = 3, 335)

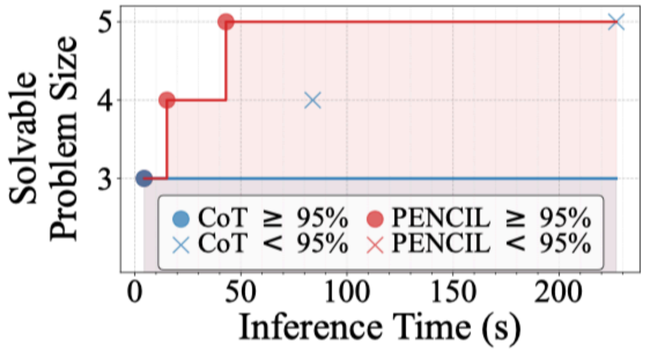
Test-Time Scalability



3-SAT



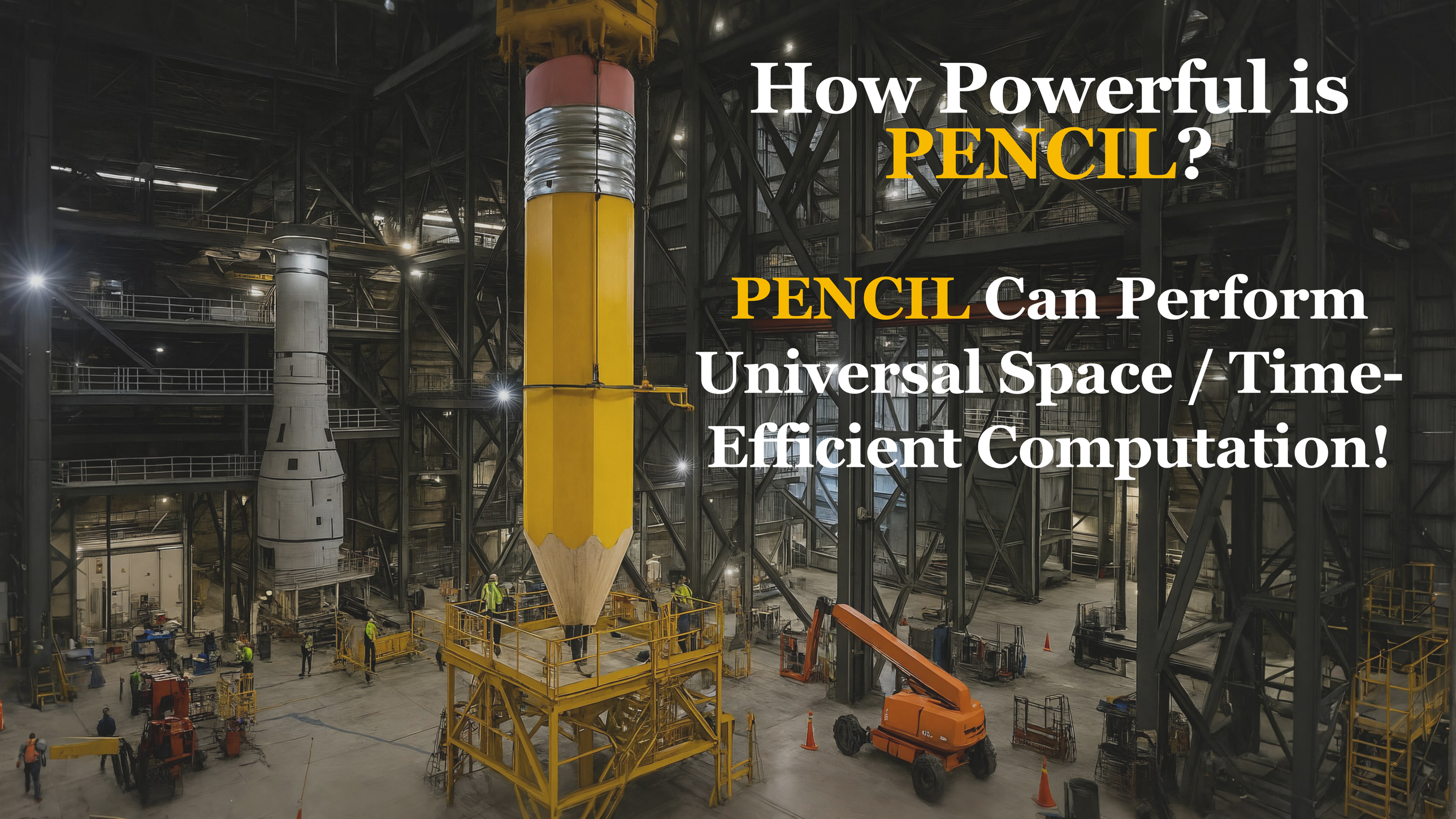
QBF



Einstein's Puzzle

Given more inference time, ✏️**PENCIL** can solve larger-sized problems.

How about other tasks beyond 3-SAT, QBF, Einstein's Puzzle?

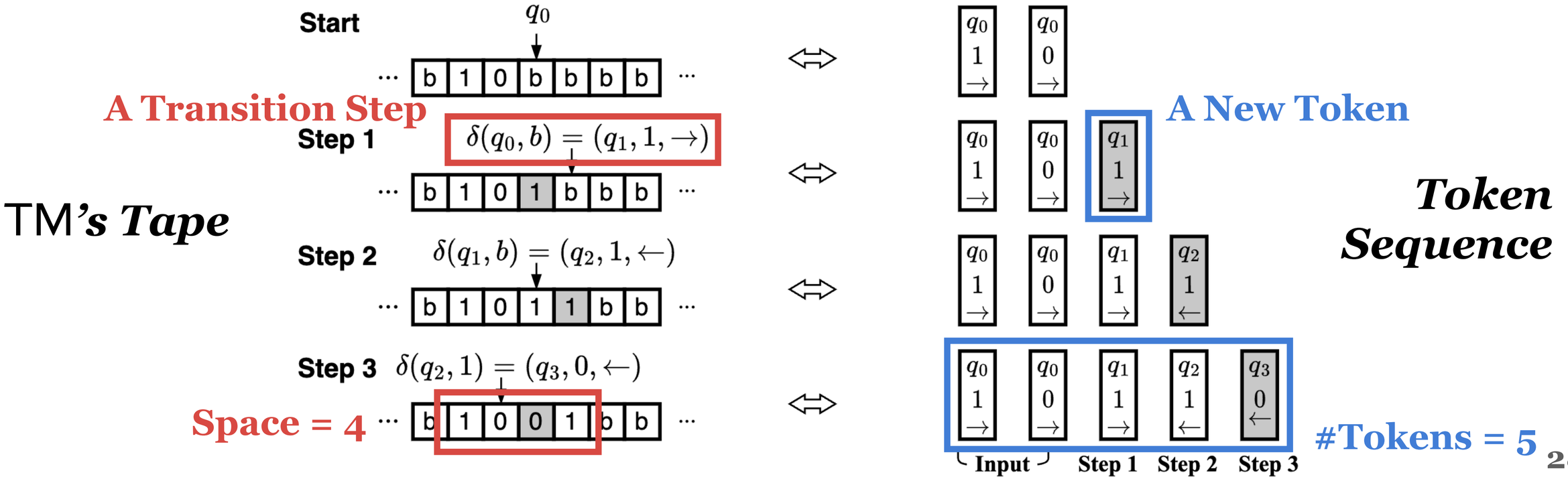

How Powerful is PENCIL?
PENCIL Can Perform Universal Space / Time-Efficient Computation!

🔪CoT is Turing-Complete, but Inefficiently

Theorem (Merrill et al. 24, Joshi at al. 25, etc.)

For any Turing machine TM, there exists a finite-size decoder-only transformer such that 🔪**CoT** with this transformer simulates the **Turing machine** with

- **Total number of generated tokens = Maximal context length =** $\mathcal{O}(T)$



Universal Efficient Computation Power of ✏️PENCIL

Theorem (Main, Informal)*

For any Turing machine, there exists a finite-size decoder-only transformer such that for any input, on which **Turing machine** uses T **steps** and S **space** to compute, ✏️**PENCIL** with this transformer computes the same output with

1. **Total number of generated tokens** = $\mathcal{O}(T)$
2. **Maximal context length** = $\mathcal{O}(S)$

- For complex problems, typically $S \ll T$
- PENCIL is Turing-complete with optimal time and space complexity
- PENCIL can solve ANY computable tasks efficiently

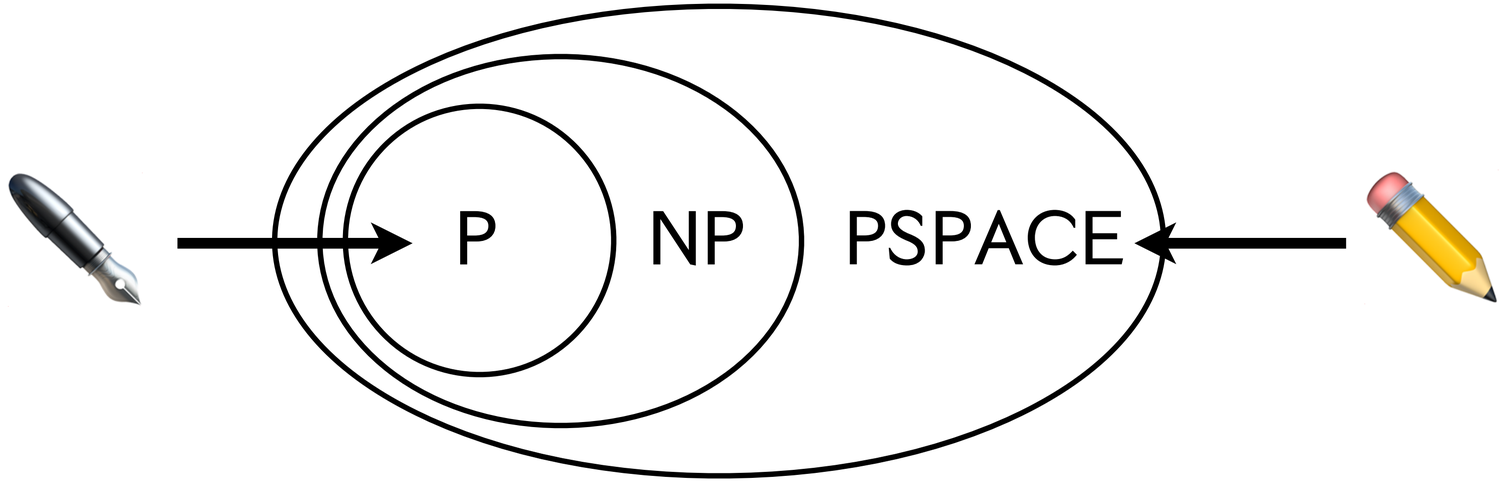
** finite size and finite parameter precision, but infinite precision in forward pass. Also assumes average-hard attention.*

Universal Efficient Computation Power of ✏️PENCIL

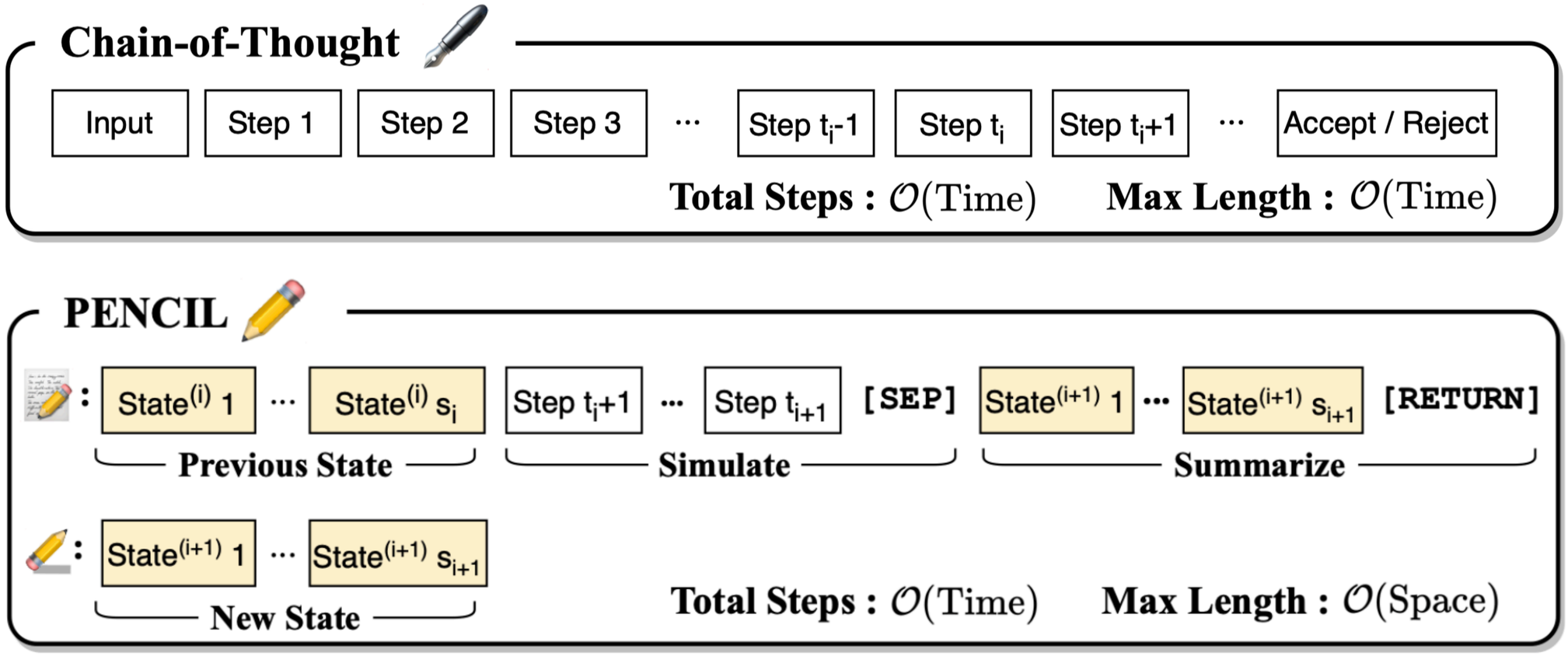
Corollary (Informal)

With poly(n) context length, ✏️**PENCIL** can solve all problems in PSPACE, while standard 🔪**CoT** can only solve problems in P.

- P : Problems solvable in **polynomial time.**
- PSPACE : Problems solvable using polynomial space, **regardless of time.**



Strategy: Iterative "Think" and "Summarize"



- Step : simulating a computation step of TM
- State : the current configuration of TM (written symbols)

When to Summarize?

Length exceeds twice the actual needed space.

When?	Never (🪶)	Every Step (✏️)	2|T'|< |T| (✏️)
# Tokens Generated	O(Time) ✅	O(Time × Space) ❌	O(Time) ✅
Max Context Length	O(Time) ❌	O(Space) ✅	O(Space) ✅

Recall we use [CALL] **T** [SEP] [CALL] **T'** [RETURN] ⇒ [CALL] **T'** *to summarize.*

BUT, can *transformers* automatically detect when to summarize / erase ?

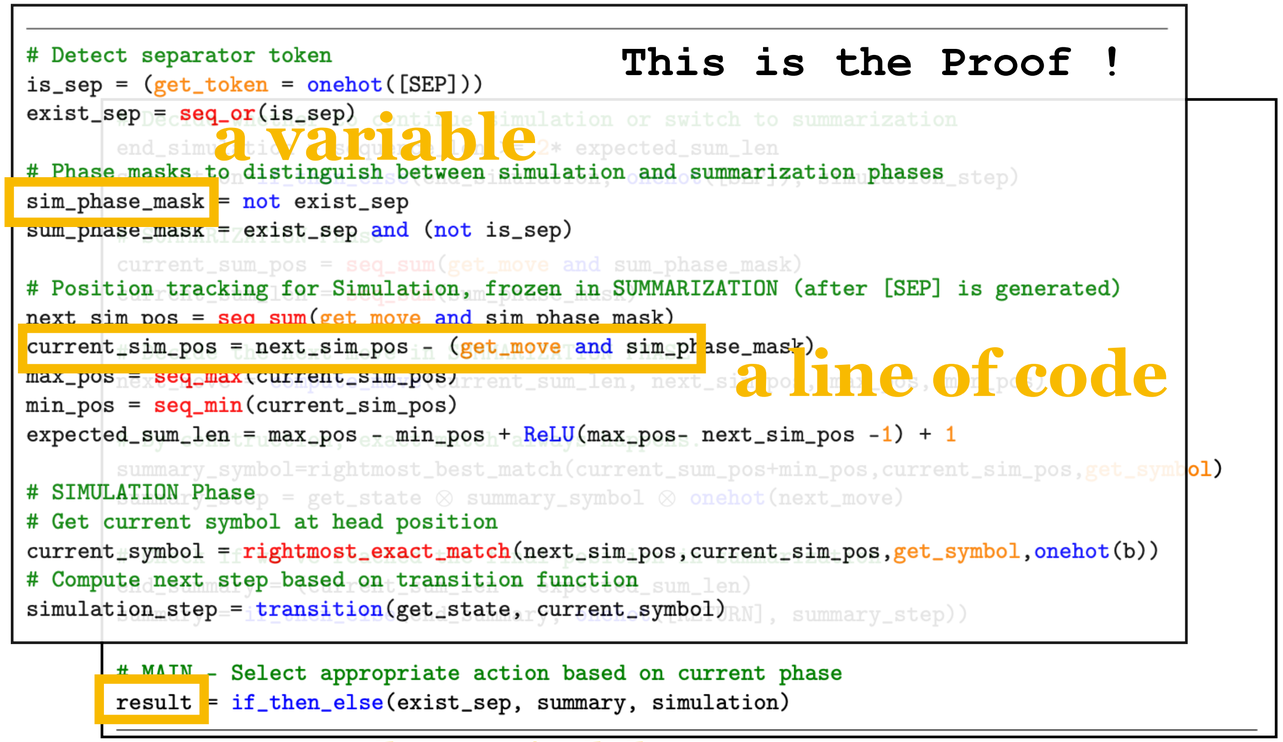
Proof Technique: FASP (Full-Access Sequence Processing)

Lemma (Informal)

Programs in FASP = All finite-size transformer functions

A FASP program describes a process of constructing transformers

- **Each Variable** = a transformer
- **Each Line of Code** = an operator from simpler transformers to a more complex transformer
- **Returned Variable** = the target transformer one aims to construct



Future Directions & Open Questions

- How to incorporate PENCIL into real-world LLM systems? How to more effectively teach them to reason in a structured way?
- Does there exist other “erasing” mechanisms (e.g. other reduction rules) that are even more efficient than PENCIL?
- Are there any other perspectives from which theories in TCS can help guide practice?

Takeaways

1. We propose ✏️ **PENCIL**, a new LLM reasoning paradigm that iteratively generates and erases thoughts using the **reduction rule**:

$$\mathbf{C}\ \texttt{[CALL]}\ \mathbf{T}\ \texttt{[SEP]}\ \mathbf{A}\ \texttt{[RETURN]} \Rightarrow \mathbf{C}\ \mathbf{A}$$

3. Empirically, ✏️ **PENCIL** enables longer and deeper thinking using shorter context, and thus can scale up to handle more complicated tasks.

4. Theoretically, ✏️ **PENCIL** is Turing-complete with optimal space and time complexity, and thus can solve arbitrary computable problems efficiently.

Thanks !