

PENCIL :

Long Thoughts with Short Memory



Chenxiao Yang, Nathan Srebro, David McAllester, and Zhiyuan Li

OptiML Group Meeting




April 4th, 2025

Presented by **Hanseul Cho**

Overview




Pen 	Pencil 

Overview

Pen 	Pencil 
NOT erasable: Write-Only	Erasable: Write  Erase
Once written, intermediate steps must remain in the scratchpad	Unless needed, intermediate steps can be erased from the scratchpad
Space-<u>inefficient</u>	Space-efficient
Error-prone due to wrong scratches	Can correct wrong scratches

Overview



Chain-of-Thought (CoT) 	PENCIL 
NOT erasable: Write-Only	Erasable: Write  Erase
Once written, intermediate steps must remain in the scratchpad	Unless needed, intermediate steps can be erased from the scratchpad
Space-inefficient	Space-efficient
Error-prone due to wrong scratches	Can correct wrong scratches

Chain-of-Thought (CoT) [Wei et al., 2022]

≡ Scratchpad [Nye et al., 2021]

- **Longer thoughts for better reasoning at inference time.**
- Originally a prompting technique for LLMs [Wei et al., 2022].
 - GPT3's Zero-shot CoT is also popular (“Let’s think step by step.” [Kojima et al., 2022])
- Most Recent LLMs are intentionally designed to do CoT (e.g., DeepSeek-R1, OpenAI o1 & o3)

Chain-of-Thought (CoT) [Wei et al., 2022]

≡ Scratchpad [Nye et al., 2021]

- **Longer thoughts for better reasoning at inference time.**
- Originally a prompting technique for LLMs [Wei et al., 2022].
 - GPT3's Zero-shot CoT is also popular (“Let’s think step by step.” [Kojima et al., 2022])
- Most Recent LLMs are intentionally designed to do CoT (e.g., DeepSeek-R1, OpenAI o1 & o3)
- Theoretically better expressivity of constant-sized, average-hardmax, [...] Transformers
 - Vanilla Transformer (no CoT): **can't** solve tasks outside of TC^0 [Merril & Sabharwal, 2023]
 - Transformer with polynomial-length CoT: **can** solve tasks in P [Merril & Sabharwal, 2024]

* $TC^0 \subset NC^1 \subset P \subset NP \subset PSPACE$. Strict (in)equalities are all open-problem.

Chain-of-Thought (CoT) [Wei et al., 2022]

What's **bad** about it? (Specifically for Transformers)

- Because of its *write-only* limitation, it often suffers from:
- 😬 Excessive **memory** resources
 - The attention layer can take quadratically growing memory in context length
- 😬 Excessive **computation** (in FLOPs)
 - Each next-token prediction step can take linearly growing computation in context length
- 😬 **Long-context reasoning is complex** in general
 - Reasoning performance tends to become worse as the context gets longer

Method: PENCIL

Pencil **EN**ables **C**ontext-efficient **I**nference and **L**earning

- **Erasable CoT: Next-Token Generator** (=pencil) + **Reduction Rule** (=eraser)
 - Add 3 *special tokens* to vocabulary: [CALL], [SEP], [RETURN]
 - **Reduction Rule:** **C [CALL] T [SEP] A [RETURN] ⇒ C A**
 - **C** for “Context” (Upcoming subproblems)
 - **T** for “Thoughts” (Intermediate steps; NOT useful for future reasoning; to be deleted)
 - **A** for “Answer” (Answer of a subproblem; useful for future reasoning)
 - To allow unique parsing, **T** can’t have [CALL]; **A** can’t have [SEP] & [RETURN].

Method: PENCIL

Pencil **EN**ables **C**ontext-efficient **I**nference and **L**earning

- **Erasable CoT: Next-Token Generator** (=pencil) + **Reduction Rule** (=eraser)
- Intuitively, the PENCIL reduction rule works GREEDILY:

- Whenever LM generates **[RETURN]**;
 1. Find the most recent **[SEP]**;
 2. Find the most recent **[CALL]**;
 3. Apply the reduction rule **C [CALL] T [SEP] A [RETURN] ⇒ C A**

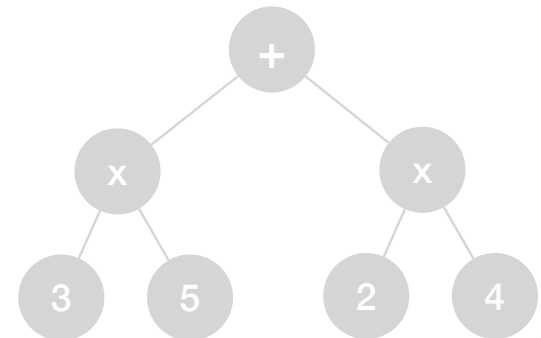
- Otherwise, keep repeating the next-token generation.
- If LM generates **[EOS]**, then halt the inference.

Example: Arithmetic with CoT

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

- **Response:**

- There are two multiplications and one addition.
- Let's follow the order of operations.

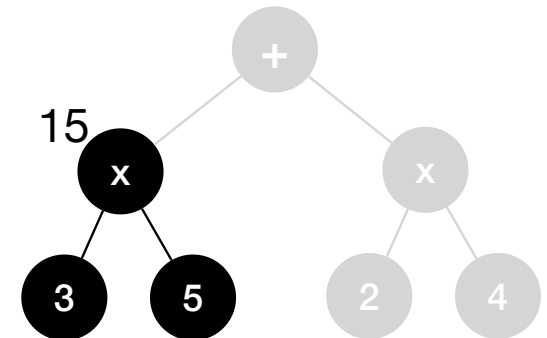


Example: Arithmetic with CoT

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

- **Response:**

- There are two multiplications and one addition.
- Let's follow the order of operations.
- Computing the first multiplication, we have $3 \times 5 = 15$.
- The first multiplication equals 15.

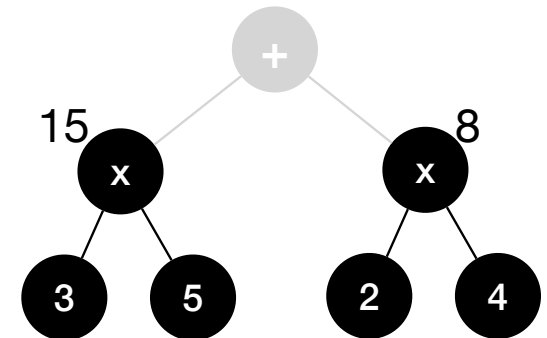


Example: Arithmetic with CoT

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

- **Response:**

- There are two multiplications and one addition.
- Let's follow the order of operations.
- Computing the first multiplication, we have $3 \times 5 = 15$.
- The first multiplication equals 15.
- Computing the second multiplication, we have $2 \times 4 = 8$.
- The second multiplication equals 8.

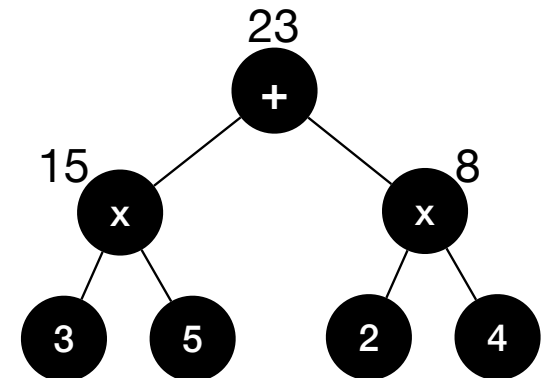


Example: Arithmetic with CoT

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

- **Response:**

- There are two multiplications and one addition.
- Let's follow the order of operations.
- Computing the first multiplication, we have $3 \times 5 = 15$.
- The first multiplication equals 15.
- Computing the second multiplication, we have $2 \times 4 = 8$.
- The second multiplication equals 8.
- Now we should add them: $15 + 8 = 23$.

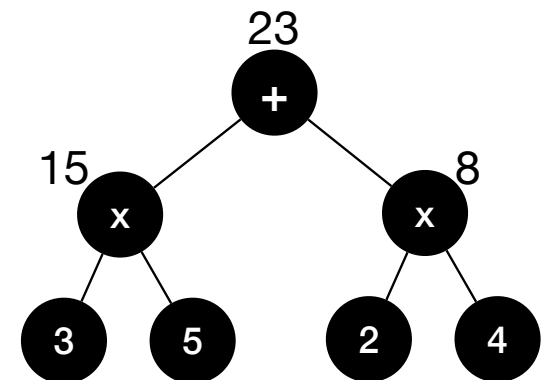


Example: Arithmetic with CoT

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

- **Response:**

- There are two multiplications and one addition.
- Let's follow the order of operations.
- Computing the first multiplication, we have $3 \times 5 = 15$.
- The first multiplication equals 15.
- Computing the second multiplication, we have $2 \times 4 = 8$.
- The second multiplication equals 8.
- Now we should add them: $15 + 8 = 23$.
- The answer is 23. [EOS]

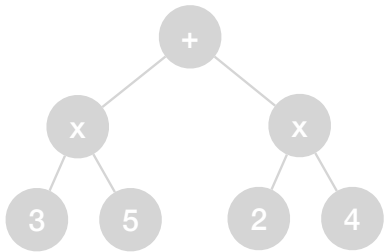


Example: Arithmetic with PENCIL

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

Yellow: Newly generated tokens

Underlined: Tokens **to be deleted** by the reduction rule




Example: Arithmetic with PENCIL

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

Yellow: Newly generated tokens

Underlined: Tokens **to be deleted** by the reduction rule

... : Prompt (omitted)

 : Generation stage

 : Reduction stage



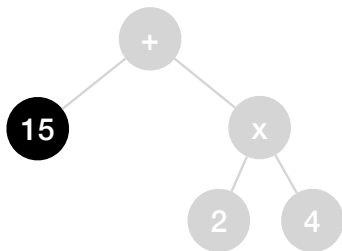
... [CALL] There are two multiplications and one addition. Let's follow the order of operations.

[CALL] Computing the first multiplication, we have $3 \times 5 = 15$.

[SEP] The first multiplication equals 15. [RETURN]



... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15.

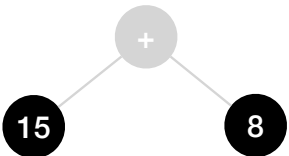


Example: Arithmetic with PENCIL

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

Yellow: Newly generated tokens
Underlined: Tokens to be deleted by the reduction rule

... : Prompt (omitted)
✍️ : Generation stage
✂️ : Reduction stage




✍️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations.
[CALL] Computing the first multiplication, we have $3 \times 5 = 15$.
[SEP] The first multiplication equals 15. [RETURN]

✂️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15.

✍️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15.
[CALL] Computing the second multiplication, we have $2 \times 4 = 8$.
[SEP] The second multiplication equals 8. [RETURN]

✂️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15. The second multiplication equals 8.




Example: Arithmetic with PENCIL

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

Yellow: Newly generated tokens

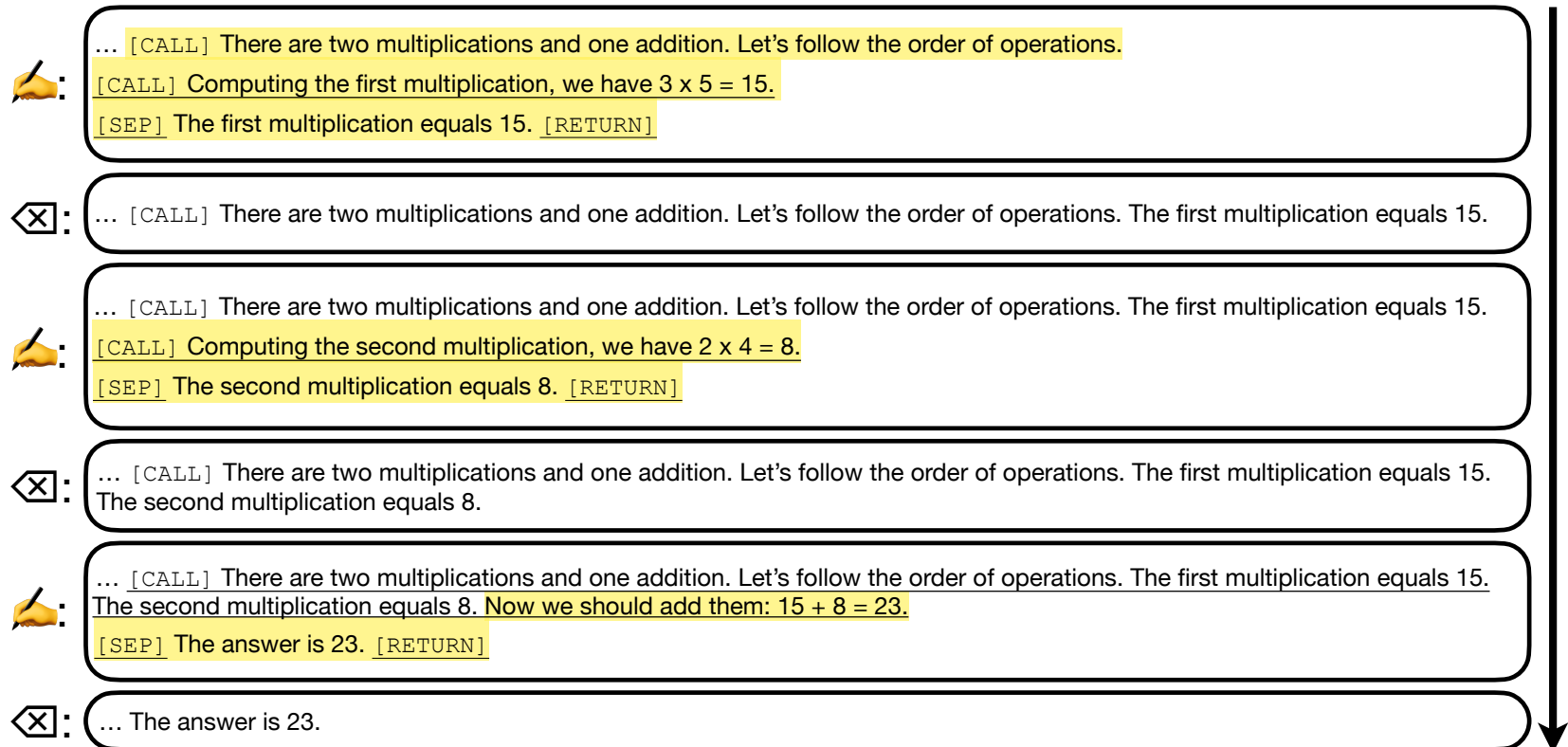
Underlined: Tokens to be deleted by the reduction rule

... : Prompt (omitted)

 : Generation stage

 : Reduction stage

23



Example: Arithmetic with PENCIL

Prompt: Compute $3 \times 5 + 2 \times 4$. [EndOfPrompt]

Yellow: Newly generated tokens
Underlined: Tokens to be deleted by the reduction rule

... : Prompt (omitted)
✍️ : Generation stage
✂️ : Reduction stage

23

✍️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations.
[CALL] Computing the first multiplication, we have $3 \times 5 = 15$.
[SEP] The first multiplication equals 15. [RETURN]

✂️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15.

✍️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15.
[CALL] Computing the second multiplication, we have $2 \times 4 = 8$.
[SEP] The second multiplication equals 8. [RETURN]

✂️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15. The second multiplication equals 8.

✍️ : ... [CALL] There are two multiplications and one addition. Let's follow the order of operations. The first multiplication equals 15. The second multiplication equals 8. Now we should add them: $15 + 8 = 23$.
[SEP] The answer is 23. [RETURN]

✂️ : ... The answer is 23.


✍️ : ... The answer is 23. [EOS]


Training Next-Token Generator for PENCIL

Objective function is slightly different from CoT

- Setting:

- $x \in \Sigma^*$: a prompt string, whose entries are elements of vocabulary Σ
- $y = \text{CoT}(x) \in \Sigma^*$: the full chain of thought (i.e., no reduction) (not including x)
- $p_\theta(\cdot | \text{context})$: a parametrized next-token generator; a distribution over vocabulary Σ
- $\phi : \Sigma^* \rightarrow \Sigma^*$: remaining string after PENCIL reduction (thus, $|\phi(c)| \leq |c|$)

-  **CoT Loss:** $L_{\text{CoT}}(x) = \sum_{i \geq 1} -\log p_\theta(y_i | x, y_{1:i-1})$ (usual loss function for causal next-token generator)

-  **PENCIL LOSS:** $L_{\text{PENCIL}}(x) = \sum_{i \geq 1} -\log p_\theta(y_i | \phi(x, y_{1:i-1}))$

- Equivalently, at every generation step, we only compute the loss for all intermediate tokens generated starting from the most recent reduction step.

Space-Efficient Universality of PENCIL

PENCIL with Transformer simulates Turing machine space-efficiently

Theorem 5.1 (Main, Informal). For any deterministic (i.e., single-tape) Turing machine TM, there exists a fixed finite-size Transformer* satisfying that:

For any input, on which the computation of TM uses T time steps and S space, the transformer using PENCIL computes the same output with $O(T)$ generated tokens and using maximal context length of $O(S)$.

“Transformer’s Architecture Choices”:

- Average-hard causal attention (using hard-max instead of softmax; no tie-breaking, but assigning uniform attention weights to every arg-max)
- Position Embedding: $n \mapsto n$
- Gated ReLU activation in FFNN, No LayerNorm...

Space-Efficient Universality of PENCIL



PENCIL with Transformer simulates Turing machine space-efficiently

Theorem 5.1 (Main, Informal). For any deterministic (i.e., single-tape) Turing machine TM, there exists a fixed finite-size Transformer* satisfying that:

For any input, on which the computation of TM uses T time steps and S space, the transformer using PENCIL computes the same output with $O(T)$ generated tokens and using maximal context length of $O(S)$.

- Remark on time/space complexity.
 - S can be $\ll T$: recall that $\text{NP} \subset \text{PSPACE}$.
 - NP: tasks verifiable (given a solution) by a deterministic Turing machine using polynomial time.
 - **Solving** an NP task may take **exponentially growing time** (probably inevitably, unless $\text{P} = \text{NP}$).
 - PSPACE: tasks solvable by a deterministic (single-tape) Turing machine using **polynomial space**.

- **Comparison with CoT.**

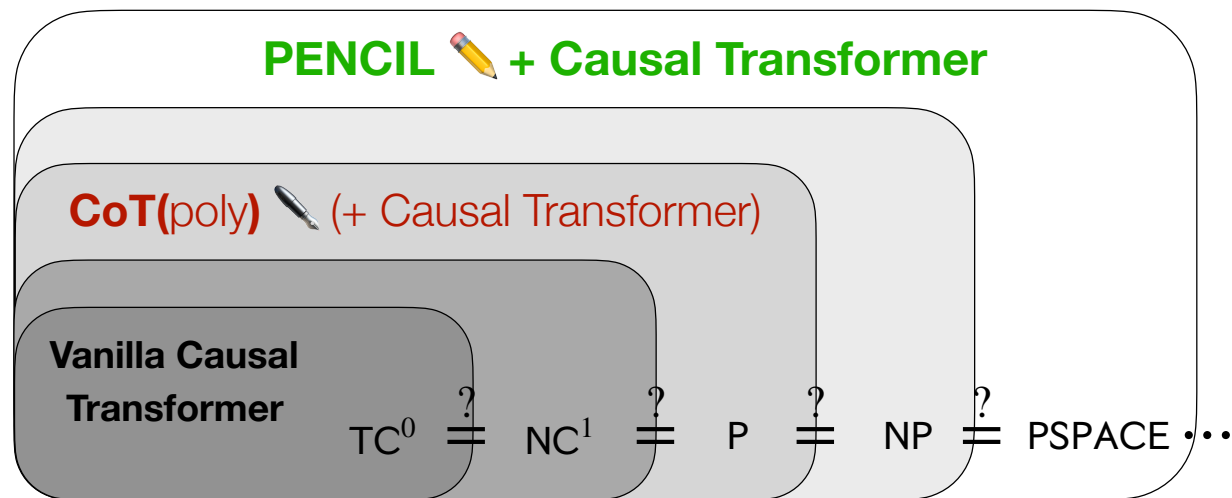
	CoT 	PENCIL 
# Tokens Generated	\approx Runtime of TM	\approx Runtime of TM
Maximal Context Length	\approx Runtime of TM	\approx Space complexity of TM

Space-Efficient Universality of PENCIL

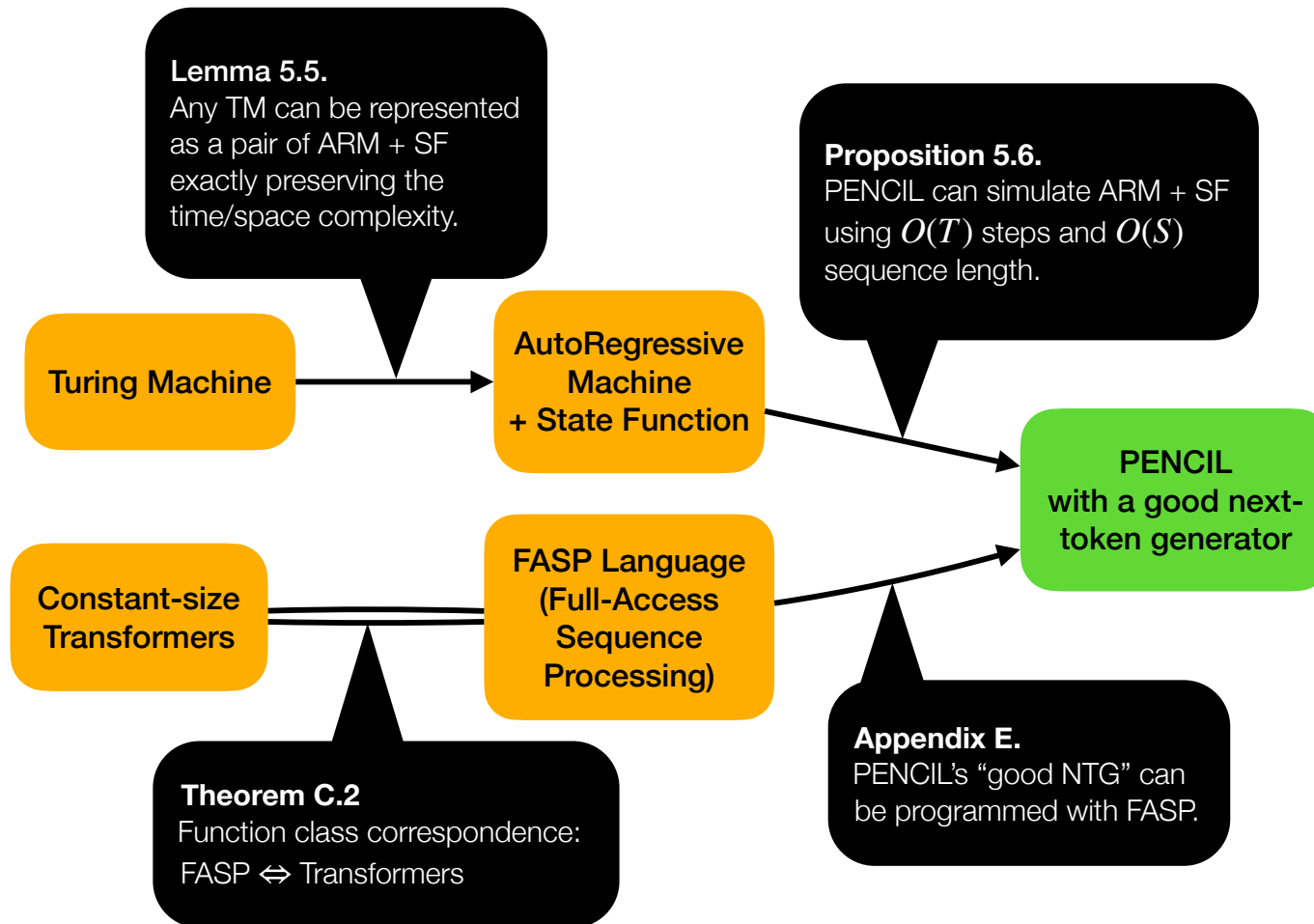
PENCIL with Transformer simulates Turing machine space-efficiently

Corollary 5.2. Assume the max context length is limited to the polynomial scale of the input prompt length. Then, Transformers with PENCIL can solve all problems in PSPACE.

However, Standard CoT with ANY poly-time next-token generator (including Transformer) can **only** solve P.



Proof in a Nutshell



Experiment: SAT & QBF

Problems with Boolean Formula

- **SAT** (Boolean formula **SAT**isfiability problem)
 - “Decide whether a formula $\phi(x_1, \dots, x_n)$ in n boolean variables is **satisfiable**, i.e., there exists an instance $\{x_i\}_{i=1}^n \in \{\text{True}, \text{False}\}^n$ so that $\phi(x_1, \dots, x_n)$ is True.”
 - This task is **NP-complete**:
 - NP because a satisfiable formula can be verified in a linear time.
 - Complete: Every NP task can be reduced to SAT in polynomial time (**Cook-Levin Theorem**).

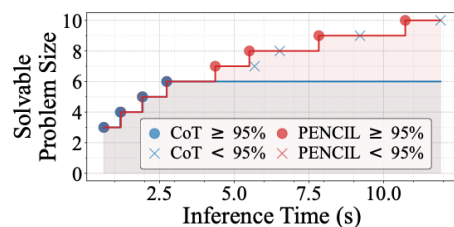
Experiment: SAT & QBF

Problems with Boolean Formula

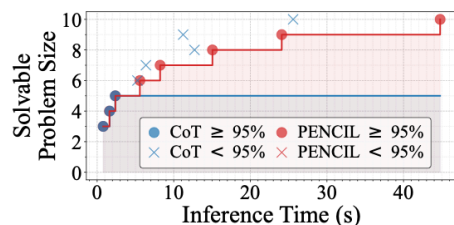
- **TQBF** (True **Q**uantified **B**oolean **F**ormula problem), or just **QBF**
 - “Decide whether a formula with quantifiers (\forall , \exists) is True.”
 - For example: “Is $\exists x_1 \forall x_2 \exists x_3 : (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$ True?”
 - SAT is a special case choosing every quantifier as existential (\exists).
 - This task is **PSPACE-complete**: (recall that $\text{NP} \subset \text{PSPACE}$)
 - May not be NP: even though we have a True QBF, it’s difficult to verify without checking almost all possible combinations of variables.
 - Complete: Every PSPACE task can be reduced to QBF in polynomial time.

Experiment: SAT & QBF

Problems with Boolean Formula



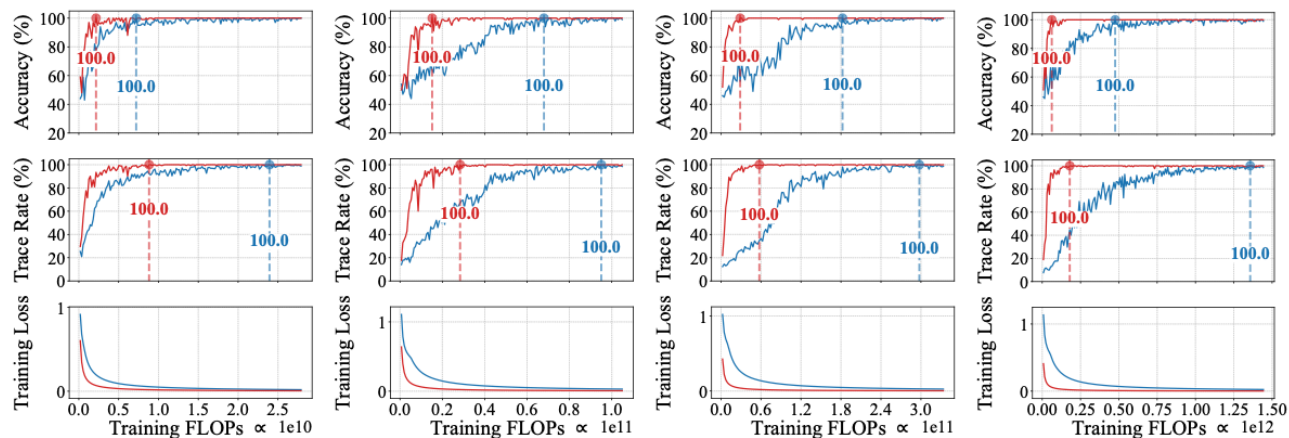
(a) SAT



(b) QBF

Figure 5. SAT & QBF, Comparison of maximally solvable problem size (with $\geq 95\%$ accuracy) given different **inference time budgets**.

- ▶ GPT2-scale, small Transformers (6-layer, 10.63M param) with RoPE, context len 2048
- ▶ “Problem size n ” = number of boolean variables in a given formula



(a) QBF $n = 3$

(b) QBF $n = 4$

(c) QBF $n = 5$

(d) QBF $n = 6$

Figure 6. Comparison of **convergence speed for training** on the QBF problem (with size ranges from 3 to 6). Circles and vertical lines indicate the first time each method reaches optimal performance. The x-axis is the FLOPs budget for self-attention. ‘Trace rate (%)’ refers to the ratio of reasoning steps matching the ground truth.

Some Questions / Limitations

- For an existing LLM, PENCIL **requires additional training**
 - LM should learn when to generate the special tokens ([CALL], [SEP], [RETURN])
 - Does it only require learning a small number of additional parameters (e.g., token embedding vectors, linear readout) while freezing the original model parameters?
 - Contrarily, CoT can be applied by prompting a well-trained LLM without further training
- Can PENCIL training be parallelized? (Not so clear; the code is not revealed yet: <https://github.com/chr26195/PENCIL>)
 - Due to the reduction rule, PENCIL training should be done chunk-by-chunk, with multiple forward passes; **seems challenging to parallelize the training fully.**
 - Training dataset can be constructed with a tree structure: Can we implement an efficient mini-batching of the tree-structured dataset?
- PENCIL works like depth-first search (DFS) for tree traversing!
 - Can other tree search methods be applied with CoT?
 - Can PENCIL do MCTS? Long-term Planning? Branch-and-Bound? ... Any other Applications?

References

- Kojima et al., Large Language Models are Zero-Shot Reasoners. NeurIPS 2022.
- Merrill and Sabharwal, The Parallelism Tradeoff: Limitations of Log-Precision Transformers. TACL 2023.
- Merrill and Sabharwal, The Expressive Power of Transformers with Chain of Thought. ICLR 2024.
- Nye et al., Show Your Work: Scratchpads for Intermediate Computation with Language Models. 2021. arXiv:2112.00114
- Wei et al., Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. NeurIPS 2022.
- **Yang et al., PENCIL: Long Thoughts with Short Memory. 2025. arXiv:2503.14337**