PENCIL: Long Thoughts with Short Memory

Zeyu Bian
zebian@ucsd.edu

University of Science and Technology of China

March 30, 2025

Content

Content

PENCIL: Long Thoughts with Short Memory

Chenxiao Yang Nathan Srebro David McAllester Zhiyuan Li
Toyota Technological Institute at Chicago
{chenxiao,nati,mcallester,zhiyuanli}@ttic.edu

Abstract

While recent works (e.g. o1, DeepSeek R1) have demonstrated great promise of using long Chain-of-Thought (CoT) to improve reasoning capabilities of language models, scaling it up during test-time is challenging due to inefficient memory usage — intermediate computations accumulate indefinitely in context even no longer needed for future thoughts. We propose PENCIL, which incorporates a reduction mechanism into the autoregressive generation process, allowing the model to recursively clean up intermediate thoughts based on patterns learned from training. With this reduction mechanism, PENCIL significantly reduces the maximal context length required during generation, and thus can generate longer thoughts with limited memory, solving larger-scale problems given more thinking time. For example, we demonstrate PENCIL achieves 97% accuracy on the challenging Einstein's puzzle — a task even large models like GPT-4 struggle with — using only a small 25M-parameter transformer with 2048 context length. Theoretically, we prove PENCIL can perform universal space-efficient computation by simulating Turing machines with optimal time and space complexity, and thus can solve arbitrary computational tasks that would otherwise be intractable given context window constraints.

Motivation

Features of **CoT** (Chain of Thoughts):

- A prompting technique to let the model generate "thinking process" before output the final answer.
- Could achieve great performance on reasoning tasks(e.g. maths,logics,code...) than without it.
- The intuitive explanation for this success is that the model could leverage the "thinking" context other than the original prompts (or with few-shot learning).
- Write-only nature, so that we need more memory and scale up test-time for complex tasks

Could we **dynamically clean up** "useless thoughts" during reasoning to promote efficient memory management and better performance?

PENCIL

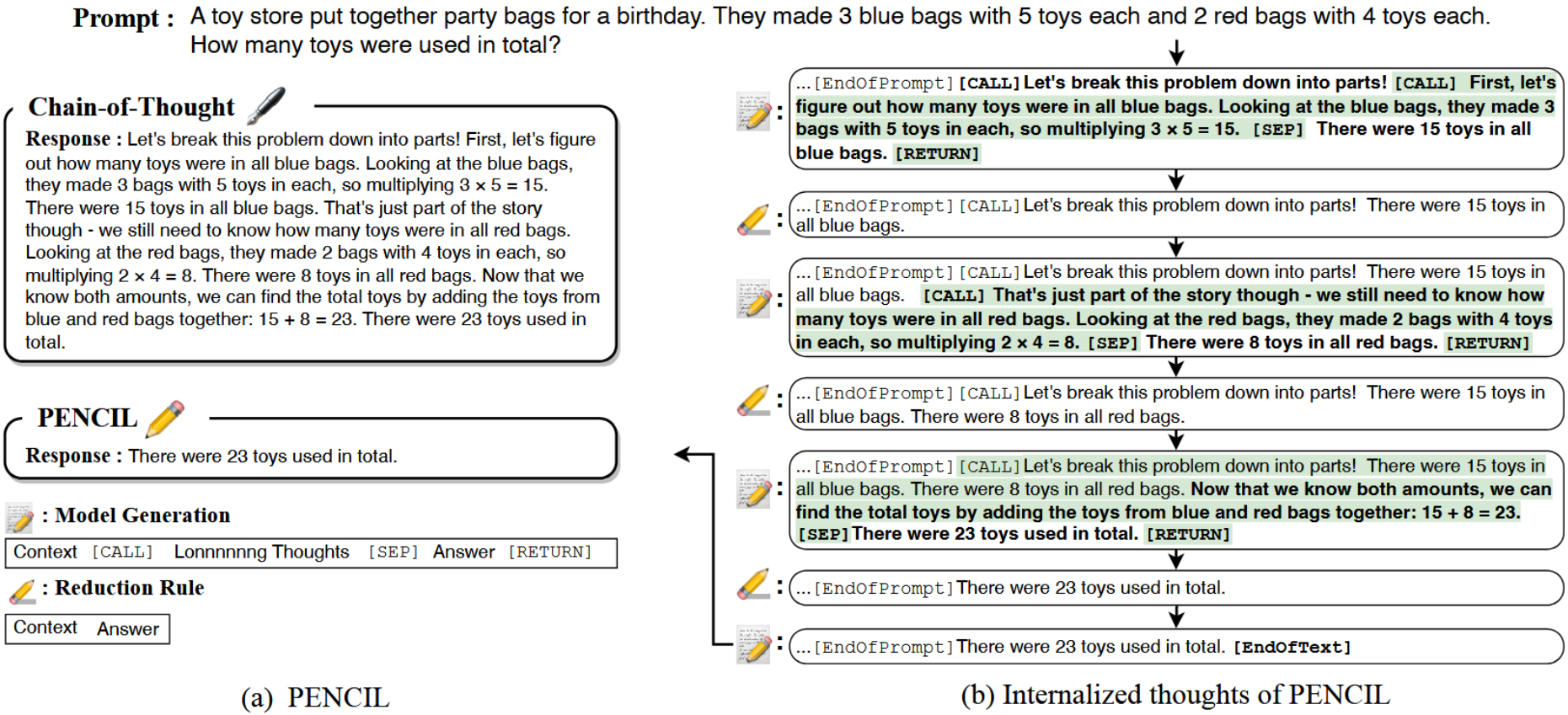
PENCIL: Combines a next-token generator (e.g. a decoder-only transformer) and a *reduction rule*, and applies the reduction rule whenever possible throughout the standard iterative next-token generation process to reduce context length.

Reduction Rule 1

$$\mathbf{C}\ [\text{CALL}]\ \mathbf{T}\ [\text{SEP}]\ \mathbf{A}\ [\text{RETURN}] \Rightarrow \underbrace{\mathbf{C}\ \ \mathbf{A}}_{\text{New Context}} \tag{1}$$

- [CALL], [SEP], and [RETURN] are special tokens that separate the context (C), thoughts (T), and answer (A) in the sequence

An Example on Arithmetic Problem



(a) PENCIL (b) Internalized thoughts of PENCIL

Figure 1: A toy example of PENCIL mechanism

Content

Mathematical Formulation

Definition (next-token predictor and seq2seq function)

Given a finite alphabet Σ,

$$\pi : \Sigma^* \to \Sigma, \quad (x_1, x_2, \cdots, x_n) \in \Sigma^n \mapsto x_{n+1} \tag{2}$$

$$f : \Sigma^* \to \Sigma^*, \quad f_\pi(x_1, \ldots, x_n) \triangleq (x_1, \ldots, x_n, \pi(x_1, \ldots, x_n)) \tag{3}$$

For brevity, write f instead of f_π when the context is clear.
K steps: $f^k : \Sigma^* \to \Sigma^*$, where $f^k \triangleq f \circ f^{k-1}, f^1 \triangleq f$
Reduction Rule:

$$\phi : \Sigma^a \to \Sigma^b (b \leq a), (x_1, \ldots, x_a) \mapsto (x_{i_1}, \ldots, x_{i_b}) \tag{4}$$

Let $\hat{\Sigma} = \Sigma \cup \{$ [CALL], [SEP], [RETURN] $\}$ be the extended alphabet

$$\begin{aligned} \mathbf{C} &\in (\Sigma \cup \{[\text{ CALL }], [\text{ SEP }], [\text{ RETURN }]\})^* \\ \mathbf{T} &\in (\Sigma \cup \{ \text{ [SEP], [RETURN] } \})^* \\ \mathbf{A} &\in (\Sigma \cup \{[\ \mathbf{CALL}\ ]\})^* \end{aligned} \tag{5}$$

Mathematical Formulation

Notice that

1. The inclusion of [CALL] in C enables nested reasoning structures critical for achieving optimal space efficiency.
2. Allowing [CALL] in A enables tail recursion optimization.

Definition (PENCIL Process)

$$\text{PENCIL}_{\phi,f}^{k} = (\phi \circ f)^{k} \tag{6}$$

Grouping the f functions that are interleaved by ineffective reduction step:

$$\text{PENCIL}_{\phi,f}^{k} = f^{k_{r+1}} \circ \phi \circ f^{k_r} \circ \phi \circ \cdots \circ \phi \circ f^{k_1} \tag{7}$$

where $k = \sum_{i=1}^{r+1} k_i$, and k_i denotes the number of tokens generated between the $(i-1)$-th and i-th effective reduction, r is the total number of effective reductions.

Mathematical Formulation

Proposition

$$\textit{Generation: } x^{(i)} \triangleq f^{k_i} \circ \underbrace{\phi \cdots \phi \circ f^{k_1}(x)}_{x^{(i-0.5)}} \tag{8}$$

$$\textit{Reduction: } x^{(i+0.5)} \triangleq \phi \circ \underbrace{f^{k_i} \cdots \phi \circ f^{k_1}(x)}_{x^{(i)}} \tag{9}$$

The complete reasoning trace:

$$x \xrightarrow{f^{k_1}} x^{(1)} \xrightarrow{\phi} x^{(1.5)} \ldots x^{(r+0.5)} \xrightarrow{f^{k_{r+1}}} x^{(r+1)} \tag{10}$$

where $x^{(i)}$ represents a generated sequence ending with [RETURN] except for $x^{(r+1)}$ which ends with the [EOS] token.

Space Efficiency

Definition (Scaffolded CoT)

without removing the thoughts:

$$\left(x, x^{(1)} \backslash x^{(0.5)}, \ldots, x^{(r+1)} \backslash x^{(r+0.5)}\right) \tag{11}$$

where $x^{(i)} \backslash x^{(i-0.5)}$ represents the tokens generated at iteration i (i.e. f^{k_i})

- The maximal sequence length in PENCIL is $\max_{i \in [r+1]} \left\{ \left| x^{(i)} \right| \right\}$, whereas the scaffolded CoT has a length of $n + k$.
- We would see their difference becomes particularly significant (i.e. $\max_{i \in [r+1]} \left\{ \left| x^{(i)} \right| \right\} \ll n + k$) for complex reasoning tasks, where the context length of CoT can grow **exponentially** while the context length length of PENCIL is kept **polynomial**.

Computational Benefits

FLOPs(Floating Point Operations): measure the total number of floating-point arithmetic operations (e.g., additions, multiplications, divisions) performed by an algorithm, model, or program.
The corresponding FLOPs for self-attention required for a problem instance $x \in \Sigma^n$ is proportional to:

$$\sum_{i=1}^{r+1} \left(\left| x^{(i-0.5)} \right| + \left| x^{(i)} \right| + 1 \right) \cdot \underbrace{\left(\left| x^{(i)} \right| - \left| x^{(i-0.5)} \right| \right)}_{\text{number of generated tokens}}$$
$$+ \sum_{i=1}^{r} \left(\left| x^{(i)} \cap x^{(i+0.5)} \right| + \left| x^{(i+0.5)} \right| + 1 \right) \cdot \underbrace{\left| x^{(i+0.5)} \backslash x^{(i)} \right|}_{\text{length of the answer A}} \tag{12}$$

where $x^{(i)} \cap x^{(i+0.5)}$ represents the shared context $\mathbf{C}$ before the [CALL] token, and $x^{(i+0.5)} \backslash x^{(i)}$ denotes the answer A between [SEP] and [RETURN] tokens.

SAT and QBF Problem

1. SAT(Boolean satisfiability problem, NP-complete): Given a Boolean formula, does there exist an assignment of values (true or false) to its variables that makes the entire formula evaluate to true?
   - **Logics**: AND(conjunction), OR(disjunction), NOT(negation)
   - **Variable**: A symbol (e.g., x_1, x_2) that can take a Boolean value: true (1) or false (0).
   - **Literal**: A variable or its negation (e.g., $x_1, \neg x_2$).
   - **Clause**: A disjunction (logical $OR, \vee$) of one or more literals (e.g., $(x_1 \vee \neg x_2 \vee x_3)$).
   - **Conjunctive Normal Form (CNF)**: A formula structured as a conjunction (logical AND, $\wedge$) of clauses. e.g.

$$\underbrace{(x_1 \vee \neg x_2)}_{\text{Clause 1}} \wedge \underbrace{(\neg x_1 \vee x_3)}_{\text{Clause 2}} \wedge \underbrace{(x_2 \vee \neg x_3)}_{\text{Clause 3}}$$

Consider the 3-SAT variant (Clause length equals three), and $\frac{n_{clauses}}{n_{variable}} = 4.3$

2. QBF(Quantified Boolean Formula, PSAPCE-complete) generalizes SAT by adding universal $\forall$ and existential $\exists$ quantifiers. Set the probability of a variable being existentially quantified as 0.5.

Algorithm for Solving SAT and QBF

Use the DPLL algorithm[Nieuwenhuis et al., 2006] to solve the SAT problem, and solving the QBF problem by recursively handling quantifiers and trying variable values.
Key features of DPLL:

- **Backtracking Search**:Systematically assigns values to variables and backtracks if a contradiction is found.
- **Unit Propagation**:Unit clause refers clause with only one unassigned literal remains, determine it for satisfiability.
- **Pure Literal Elimination**: A pure literal is a variable that appears only in positive form (x_1), or only in negative form($\neg x_1$). Assign the same value.

DPLL Pseudocode

```
Algorithm DPLL
    Input: A set of clauses Φ.
    Output: A truth value indicating whether Φ is satisfiable.
```

```
function DPLL(Φ)
    // unit propagation:
    while there is a unit clause {l} in Φ do
        Φ ← unit-propagate(l, Φ);
    // pure literal elimination:
    while there is a literal l that occurs pure in Φ do
        Φ ← pure-literal-assign(l, Φ);
    // stopping conditions:
    if Φ is empty then
        return true;
    if Φ contains an empty clause then
        return false;
    // DPLL procedure:
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {¬l});
```

Figure 2: DPLL_Pseudocode

SAT and QBF Results

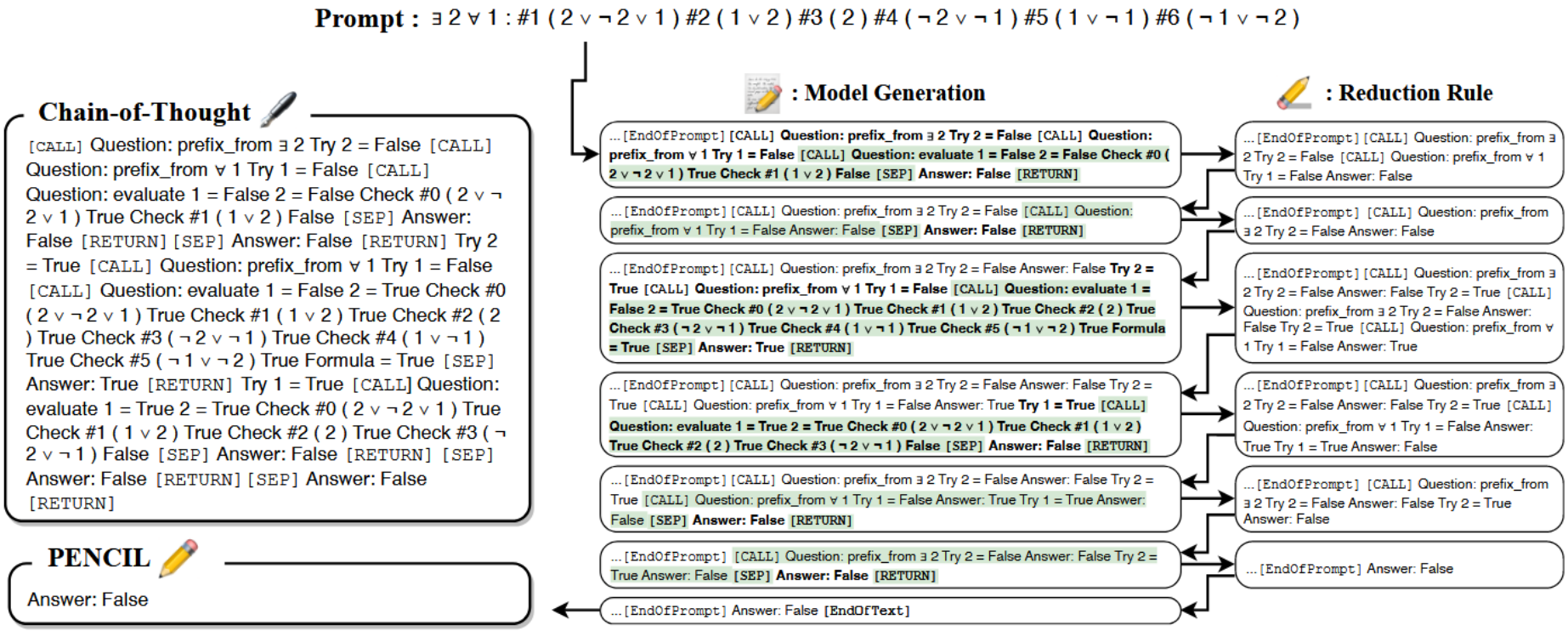


Figure 3: The complete thinking process of PENCIL on a small-sized QBF

- Unassigned variable x_i: trying branches x_i = True and x_i = False. The reduction rule wraps each branch, thus creates a **hierarchical binary tree** structure.

SAT and QBF Results: Sequence Length

- Without reduction: leading to worst-case exponential space complexity $\mathcal{O}(2^n)$.
- For PENCIL, this reduces the maximal length to $\mathcal{O}(n)$, bounded by the search tree depth.

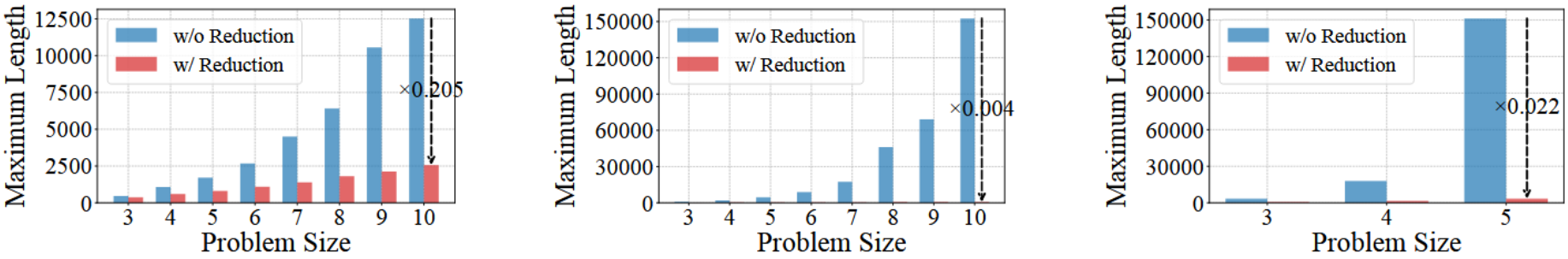


Figure 4: Maximal sequence length with and without the reduction rule

Tail Recursion and Einstein's Puzzle

- Einstein's Puzzle: Determine the attributes of each house through logical deduction. The original puzzle has size 5×5 (5 houses and 5 attribute categories, totaling 25 variables.
- Special Use Case: Tail Recursion: when $\mathbf{A} = \text{[CALL]}\ \mathrm{T}'$, (1)becomes

$$\mathrm{C}\ \text{[CALL]}\ \mathrm{T}\ \text{[SEP]}\ \text{[CALL]}\ \mathrm{T}'\ \text{[RETURN]}\ \Rightarrow \mathrm{C}\ \text{[CALL]}\ \mathrm{T}' \quad (13)$$

- It mimics the tail recursion in functional programming where a function's returned value is another function call.

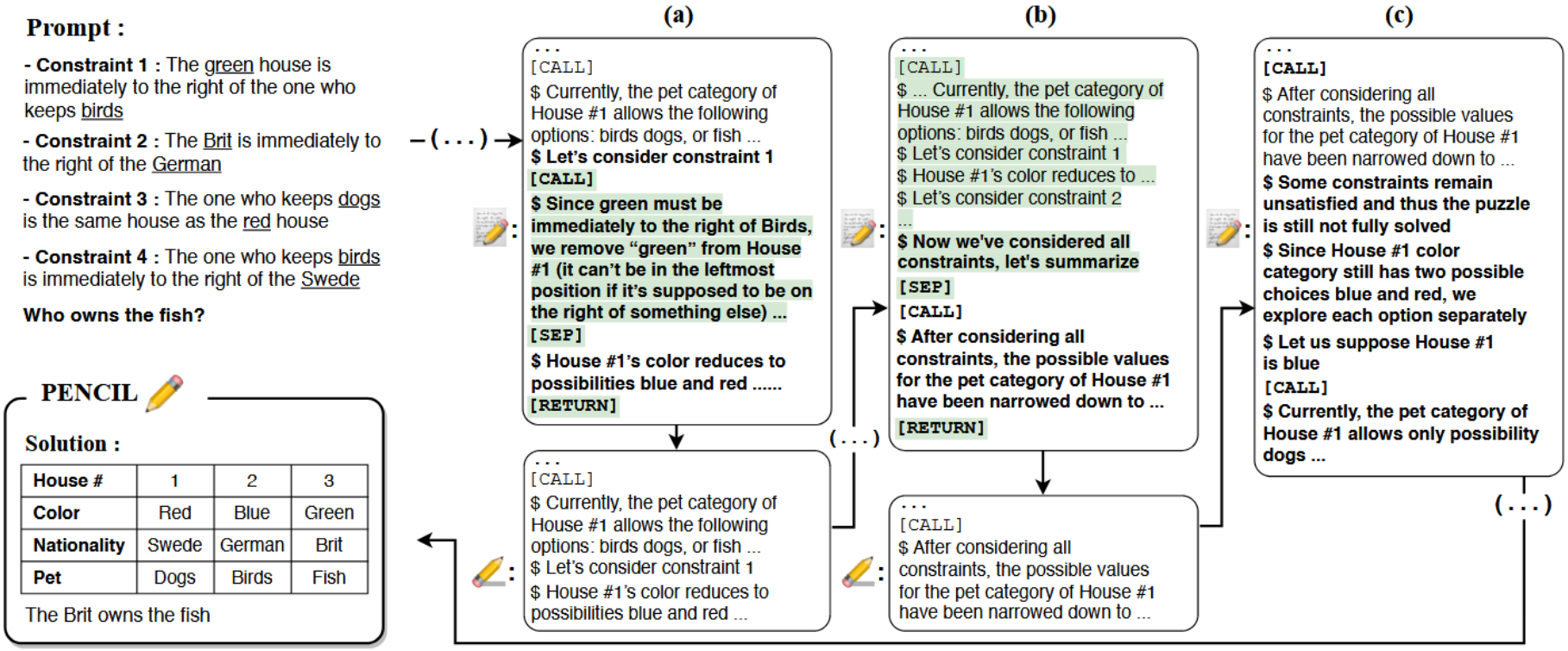


Figure 5: The thinking process for Einstein's puzzle (3×3).

Procedure:

1. Propagating constraints to eliminate impossible combinations
2. Use the tail recursion rule to merge results from constraints propagation and update the house states;
3. Iteratively explore different solution branches, only preserving the final answer.

Experiments Details

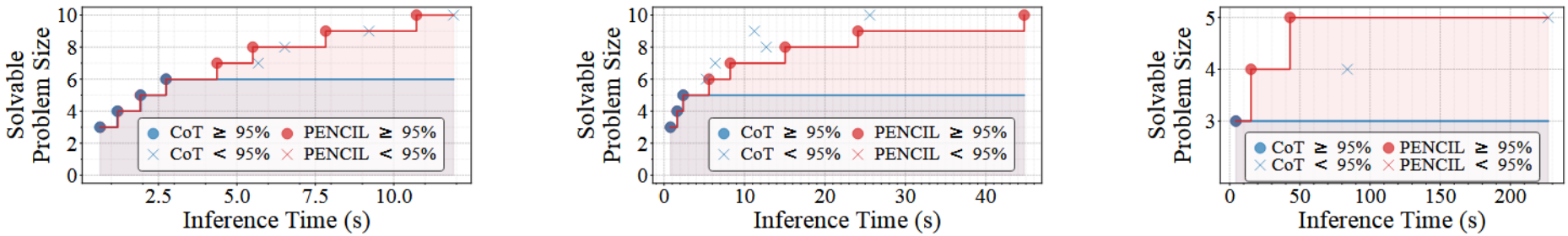
- Training:
  1. Data: Generates the corresponding scaffolded CoT (3) with special tokens as we run the algorithm, and then transform the long scaffolded CoT sequence into a set of smaller sequences $\left\{x^{(1)}, x^{(2)}, \ldots, x^{(r+1)}\right\}$ that ends with either [RETURN] or EOS.
  2. Loss: Compute loss only on generated tokens $x^{(i)} \backslash x^{(i-0.5)}$ (i.e. f^{k_i})
     Optimization:
     1. All shorter sequence from one instance into one batch
     2. Or sample sequences from all problem instances.
- Model structure:
  - SAT,QBF:6-layer transformer, 10.63M parameters
  - Einstein's puzzle: 8-layer transformer, 25.19M parameters
- Evaluation metric:
  - Accuracy: percentage of correct predictions
  - Trace rate: percentage of reasoning steps matching the ground truth(computed by the algorithm?)

SAT and QBF Results

	$n=$	3	4	5	6	7	8	9	10
Baseline	Acc.	66	57	46	51	46	51	49	51
CoT	Acc.	100	100	100	99	84	63	54	50
	TR.	99.6	99.0	98.0	96.2	74.0	69.9	63.8	51.4
PENCIL	Acc.	100	100	100	99	99	100	100	100
	TR.	100	99.0	97.1	95.9	91.8	93.3	92.9	83.0

	$n=$	3	4	5	6	7	8	9	10
Baseline	Acc.	90	82	85	68	60	69	71	66
CoT	Acc.	100	100	97	94	74	72	69	73
	TR.	100	100	98.3	93.9	65.1	49.4	40.7	32.8
PENCIL	Acc.	100	100	100	100	100	100	100	100
	TR.	100	100	100	100	100	100	100	100

Table 1: Performance comparison on SAT (left) and QBF (right). Acc denotes the Accuracy (%) and TR denotes the trace rate (%).



(a) SAT (b) QBF (c) Einstein's puzzle

Figure 6: Comparison of maximally solvable problem size (with $\geq$ 95% accuracy) given different inference time budgets.

Training Loss Example

The space is too limited here for a demonstration

Refer to page 52 Internal Thinking Process of PENCIL for demonstration on a concrete example.(Zotero)

SAT and QBF Results

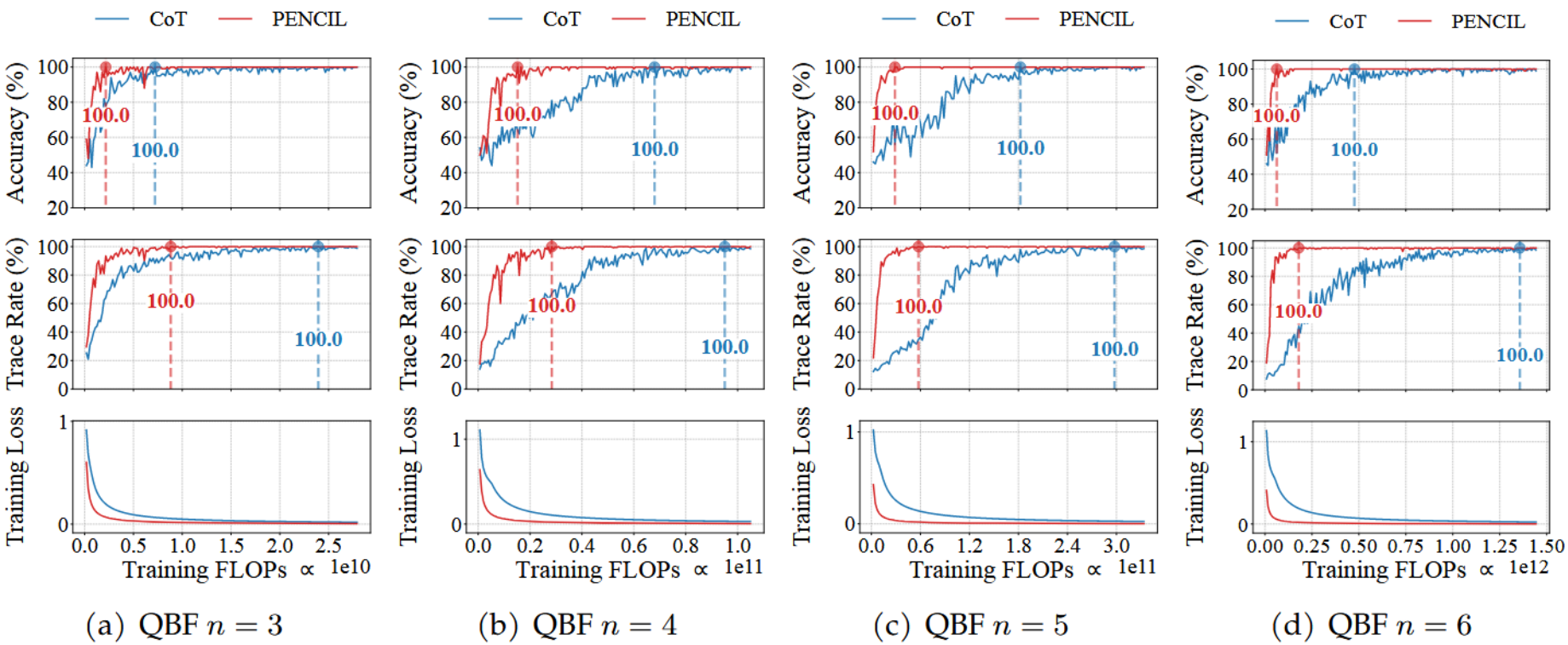


Figure 7: Comparison of convergence speed for training on the QBF problem (with n ranges from 3 to 6). Circles and vertical lines indicate the first time each method reaches optimal performance

Test Time Scalability

Test Time Scalability refers to a system's ability to efficiently handle **increased demands during the inference (prediction) phase** of a machine learning model, such as higher request volumes, larger input sizes, or more complex computations, without degrading performance.

We could see from (6) that PENCIL can effectively solve larger problems with increased time budget, possibly due to less requirement of space.

Einstein's Puzzle Results

Puzzle Size		CoT	PENCIL
5×5	Accuracy (%)	25	97
	Trace Rate (%)	2.97	78.27
4×4	Accuracy (%)	34	100
	Trace Rate (%)	8.33	86.52
3×3	Accuracy (%)	99	99
	Trace Rate (%)	99.37	99.66

Table 2: Comparison of performance w/o and with the reduction rule on Einstein's puzzle.

Einstein's Puzzle Results

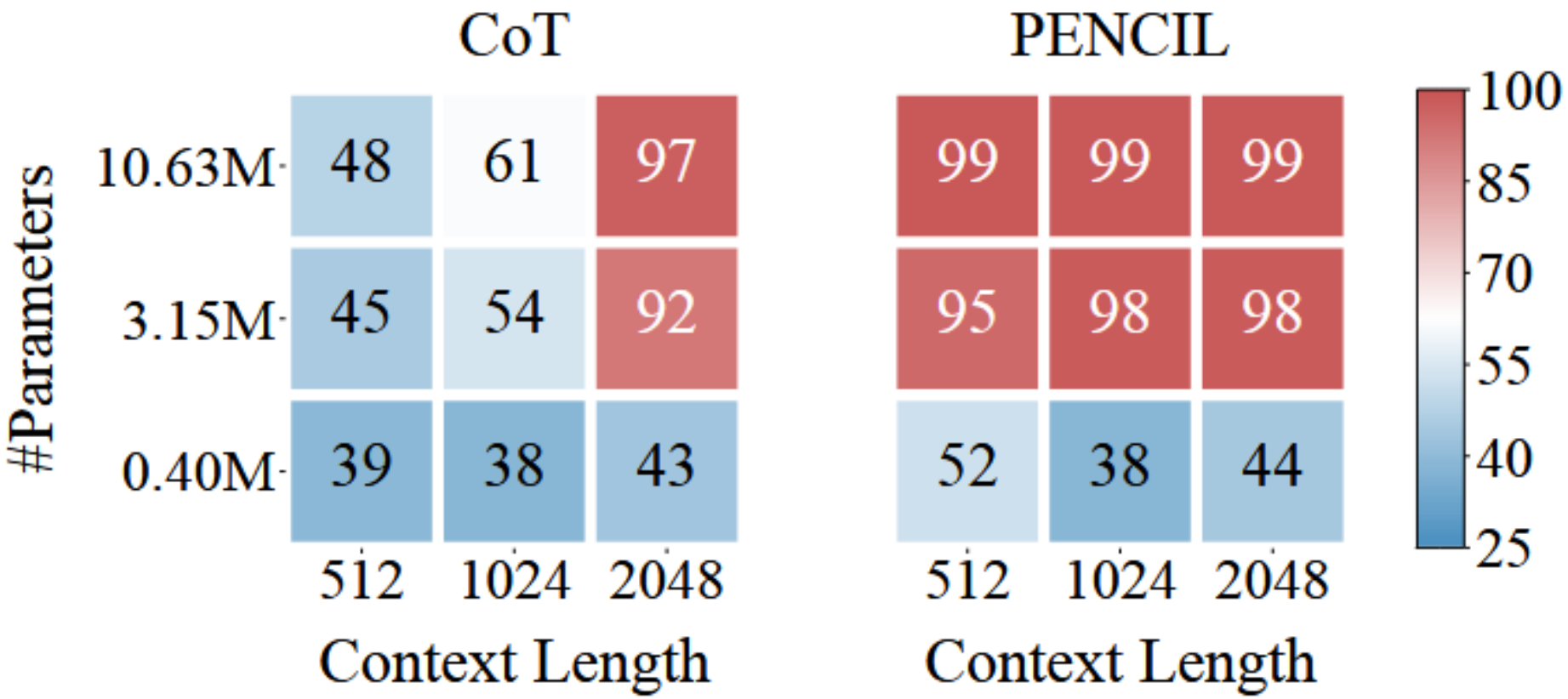


Figure 8: Effects of model size and context length on accuracy for 3×3 Einstein's puzzle.

Content

Turing Machine

Definition (Turing Machine)

A single-tape Turing machine works on a infinitely long "Tape" on both of its ends with cells indexed by integers $\mathbb{Z}$. It is specified by a 7 -tuple

$$\mathrm{TM} = (\mathcal{A}, b, Q, q_0, \delta, Q_{\text{accept}}, Q_{\text{reject}}) ,$$

where:

- $\mathcal{A}$ is a finite tape alphabet.
- $b \in \mathcal{A}$ is the designated blank symbol.
- Q is a finite set of control states.
- $q_0 \in Q$ is the initial control state.
- $\delta : Q \times \mathcal{A} \to Q \times (\mathcal{A} \backslash \{b\}) \times \{-1, 0, 1\}$ is the transition function.
- $Q_{\text{accept}} \subseteq Q$ is the set of accepting states.
- $Q_{\text{reject}} \subseteq Q$ is the set of rejecting states, disjoint from Q_{accept} .

Computation Theory

- P: Problems solvable by a deterministic Turing machine in polynomial time.
- NP: Decision problems where a "yes" answer can be **verified** in polynomial time
- PSPACE: Problems solvable with polynomial space (memory).
- EXPTIME: Problems solvable in exponential time $O(k^n)$

Task Difficulty

$P \subset NP \subset PSPACE \subset EXPTIME$
widely-conjuctured that $P \subsetneq PSAPCE$ (weaker than $P \neq NP$)
Recap: SAT,Einstein(NP-complete),QBF(PSPACE-complete)

NP-complete(The hardest problem in NP):

- The problem is in NP
- Any NP task could be reduced to it in polynomial time.

Exponential Time Hypothesis

Exponential Time Hypothesis: It is a **conjecture** that there is no algorithm that can solve the 3-SAT problem in **subexponential time** in the **worst case**.

- If ETH is true, it could suggest that P does not equal NP, supporting the idea that there are inherently hard problems within NP that cannot be solved efficiently.
- It connects deeply with the study of approximation algorithms, as it suggests that finding approximate solutions might be more feasible than exact solutions for NP-complete problems.

Autoregressive Machine

Definition (Autoregressive Machine)

It is a tuple $\mathcal{M} = (\Sigma, \pi, \Sigma_{\text{accept}}, \Sigma_{\text{reject}})$, where

- Σ is a finite alphabet
- $\pi : \Sigma^* \to \Sigma$ is a next-token generator
- $\Sigma_{\text{accept}}, \Sigma_{\text{reject}} \subseteq \Sigma$ are disjoint sets of accepting and rejecting tokens.

For any input $x \in \Sigma^*$, $\mathcal{M}$ iteratively generates one token per step and appends it to the current sequence, with $f_\pi^k(x)$ denoting the sequence after k iterations where $f_\pi(x) = (x, \pi(x))$. The machine halts when it generates a token in Σ_{accept} or Σ_{reject}.

Definition (State Function)

A function $s : \Sigma^* \to \Sigma^*$ is a state function of a autoregressive machine $\mathcal{M} = (\Sigma, \pi, \Sigma_{\text{accept}}, \Sigma_{\text{reject}})$ if (1) $\pi \circ s = \pi$; (2) for all $x, x', y \in \Sigma^*, s(x) = s(x') \Longrightarrow s((x, y)) = s((x', y))$; (**3**)$s^2 = s$

State Function

Notice that $\pi^k(x)$ for $k = 1, 2, \ldots$, can be uniquely determined by the state function s of $\mathcal{M}$:

$$s \circ f_\pi^k \circ s = s \circ f_\pi^k \text{ and } \pi^{k+1} = \pi^{k+1} \circ s \text{ for any } k \geq 0 \tag{14}$$

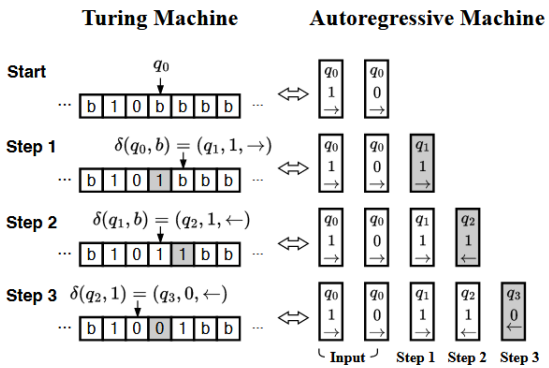
Key point: s defines a equivalent class over all possible computational traces of $\mathcal{M}$, where the mapping $x \mapsto s(x)$ erases irrelevant information while preserving the **essential information** for future computation.

Turing Machine as Autoregressive Machine

Lemma

Any Turing machine TM can be represented as a autoregressive machine $\mathcal{M}_{\mathrm{TM}}$ associated with a state function s_{TM} that preserves its time and space complexity.

Specifically, the time complexity of $\mathcal{M}_{\mathrm{TM}}$ equals the Turing machine's total step count, as each transition corresponds to exactly one token generation. The state function s_{TM} transforms the full trace into a minimal trace that contains only the current non-blank tape contents and head position, and thus the space complexity of $\mathcal{M}_{\mathrm{TM}}$ matches the Turing machine's actual memory usage.



Construction of State Function

Definition (State Function s_{TM})

Let $\mathcal{M}_{\mathrm{TM}} = (\Sigma, \pi, \Sigma_{\text{accept}}, \Sigma_{\text{reject}})$ be the autoregressive machine representation of Turing machine from Definition A.6. We define its state function $s_{\mathrm{TM}} : \Sigma^* \to \Sigma^*$ as the following

$$s_{\mathrm{TM}}(x) = \text{embed}\left(\text{Update}\left(c_0, x\right)\right), \quad \forall x \in \Sigma^*,$$

where $c_0 = \left(q_0, b^{\mathbb{Z}}, 0\right)$ is the initial configuration.

Verify that is satisfies the three requirement: Next-Token Preservation,Future-Trace Preservation,Idempotence.

Space and Time Analysis

Reduction Rule 2(Simplified)

$$\phi' : \quad \mathbf{T} \left[\text{SEP} \right] \mathbf{T}' \text{ [RETURN]} \Rightarrow \mathbf{T}' \tag{15}$$

simply set $\mathbf{T}' = s(\mathbf{T})$, the question remains as to when to generate [SEP] and trigger the reduction.

- **Space-Efficient but Time-Inefficient Solution**: trigger the summarization after every new token generation $\Rightarrow$ space $O(S)$, blow up the time.

Space and Time Efficient Solution

Trigger the summarization only when the length of T exceeds a certain **threshold**:
PENCIL compresses the current sequence into its state representation whenever its length exceeds **twice the state length**, enforcing space stays within $O(S)$ without performing reductions so frequently that the overall time cost exceeds $O(T)$.

$$f_\pi^{t_i - t_{i-1}} \circ s \circ f_\pi^{t_{i-1}}(x), \quad [\mathrm{SEP}], s \circ f_\pi^{t_i}(x), \quad [\mathrm{RETURN}] \tag{16}$$

where $s \circ f_\pi^{t_i}(x)$ is equivalent to $s \circ f_\pi^{t_i - t_{i-1}} \circ s \circ f_\pi^{t_{i-1}}(x)$

Space and Time Efficient Solution

Proposition

For any autoregressive machine $\mathcal{M} = (\Sigma, \pi, \Sigma_{accept}, \Sigma_{reject})$ *with state function* s*, if a next-token predictor* f_{π_θ} *accurately generates the next token in* (16) *from the prefix for every* i *on any input* $x \in \Sigma^*$*, then* $\text{PENCIL}_{f_{\pi_\theta}, \phi'}$ *can simulate* $\mathcal{M}$ *by using* $\mathcal{O}(T(\mathcal{M}, x))$ *steps and a maximal sequence length of* $\mathcal{O}(S(\mathcal{M}, s, x))$*.*

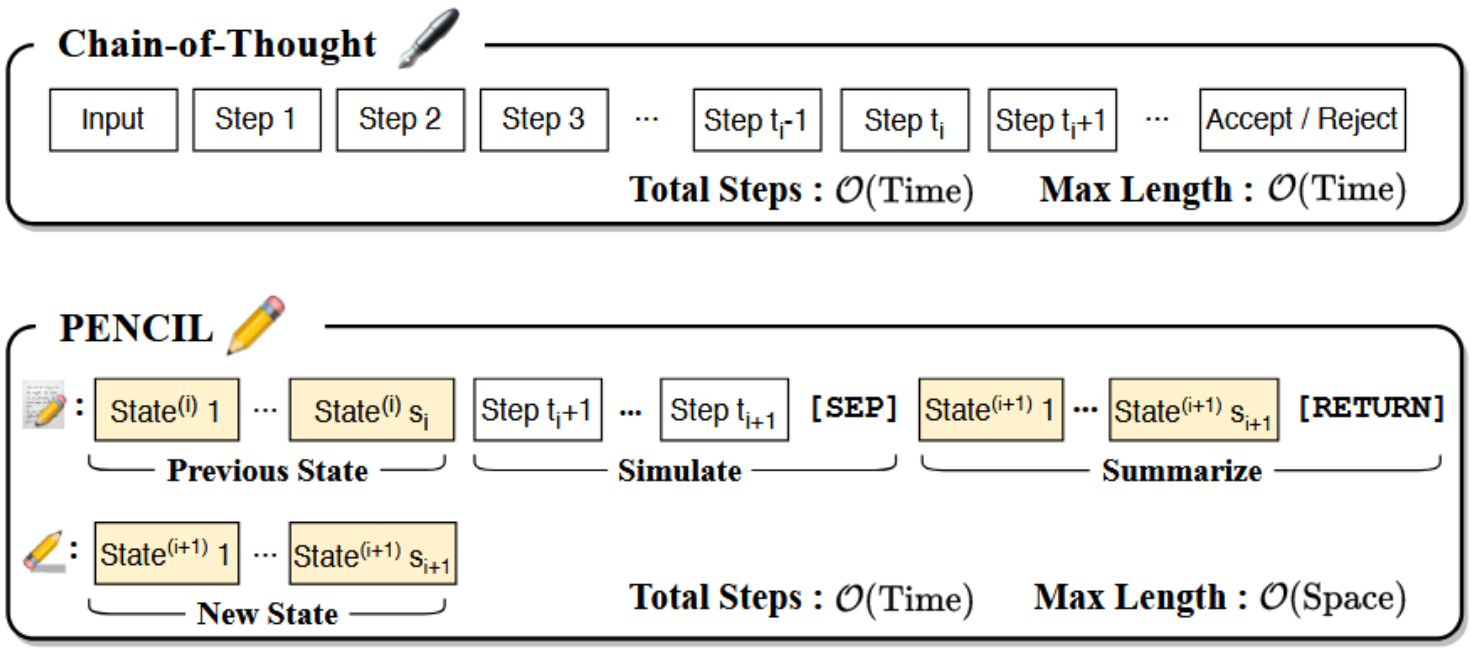
This result could applie to any computational model representable as an autoregressive machine with **a suitable state function**, i.e., whenever one can transform the full sequence into a sequence that accurately reflects the model's actual needed space.

Refer to appendix for proof.

Main Theoretical Result

Theorem (Main, Informal)

For any Turing Machine, there exists a fixed finite-size transformer such that for any input, on which the computation of Turing Machine uses T steps and S space, PENCIL with this transformer computes the same output with $\mathcal{O}(T)$ generated tokens and using maximal context length of $\mathcal{O}(S)$.



FASP

Instead of directly giving the construction of the weight matrix of each layer of the transformer, they develop a new programming language, FASP, which has the same expressiveness.

Definition (FASP)

Let $\phi_{\mathrm{PE}} : \mathbb{N}^+ \to \mathbb{R}^{\mathrm{PE}}$ be a feature function for positional embedding and $\mathcal{T}_{\mathrm{ACT}}$ be the class of activation functions. Define the FASP $[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}]$ program as the process of defining a sequence of token-sequence-to-embedding $\psi_1, \ldots, \psi_n \in \mathcal{H}$ using FASP $[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}]$ operators. The program is defined as follows: at each step $t \in [n]$, the program maintains a set of defineable seq-to-embedding functions $\mathcal{S}_t$, and defines a new function by concatenation functions in $\mathcal{S}_t$, or applying local operators (corresponding to MLP), or non-local operators (corresponding to average-hard attention) to some function in $\mathcal{S}_t$. Finally we add the newly defined function to $\mathcal{S}_t$, which yields $\mathcal{S}_{t+1}$. In detail, we define the defineable functions at step $t \in [n]$:

$$\mathcal{S}_t \triangleq \mathcal{H}_{\mathrm{TE}} \cup \{\phi_{\mathrm{PE}}\} \cup \{\psi_i \mid 1 \leq i \leq t-1\}$$

Note this also implies that $\mathcal{S}_t = \mathcal{S}_{y-1} \cup \{\psi_t\}$.

FASP Operators

1. **Concatenation:** $\psi_t = [\psi, \psi']$, where $\psi, \psi' \in \mathcal{S}_t$. This operator concatenates the output embedding vector of two functions into a longer vector.
2. **Average-Hard Attention:** $\psi_t = \text{aha}(\psi, \psi', \psi'')$, where $\psi, \psi', \psi'' \in \mathcal{S}_t$ and ψ, ψ' have the same output dimension. This implements average-hard attention with query ψ, key ψ', and value ψ''.
3. **Linear Projection:** $\psi_t = \phi \circ \psi$, where $\psi \in \mathcal{S}_t$ and ϕ is a linear transformation with arbitrary output dimension.
4. **Nonlinear Activation**: $\psi_t = \phi \circ \psi$, where $\phi : \mathbb{R}^k \to \mathbb{R} \in \mathcal{T}_{\text{ACT}}, \psi \in \mathcal{S}_t \cap \mathcal{H}(k)$ for some positive integer k.[8]

FASP Output

The final output is a function mapping from a sequence of tokens Σ^* to a single token in Σ, by returning the index with the largest value in the last function defined:

$$\boldsymbol{x} \in \Sigma^* \mapsto y = (\Sigma)_{index(x)}, \quad \text{where } index(x) = \underset{i}{argmax}\ (\psi_n)_i \tag{17}$$

Here we additionally require the (final seq-to-embeding function) to be of dimension $|\Sigma|$ and we assume an implicit order over Σ so the index maps to a token in Σ.

We denote the set of all such final outputed seq-totoken functions defineable by FASPwith position embedding ϕ_{PE} and activation functions $\mathcal{T}_{\mathrm{ACT}}$ as FASP $[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}]$.

FASP Expressiveness

Theorem (FASP Expressiveness)

For any positional encoding feature function ϕ_{PE} and activation function class $\mathcal{T}_{\mathrm{ACT}}$, it holds that FASP $[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}] = \mathcal{H}_{\mathrm{TF}[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}]}$.

The high-level idea towards the proof is to show that the four operators that generates new functions in FASP $[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}]$ are also closed under the class of embedding functions that can be implemented by transformers, namely $\mathcal{H}_{\mathrm{TF}[\phi_{\mathrm{PE}}; \mathcal{T}_{\mathrm{ACT}}]}$.

Theorem (Hierarchy of FASP Variants)

The following containment relations hold between variants of FASP:

$$\mathrm{FASP}\,[0; [\cdot]_+] \subseteq \mathrm{FASP}\,[0; [\cdot]_+, \times] \subseteq \mathrm{FASP}\,[\ \textit{is\_first}\ ; [\cdot]_+, \times] \subseteq \mathrm{FASP}\,[\ \textit{seq\_len} \tag{18}$$

where each inclusion represents a strict increase in expressiveness.

Main Results

Theorem (Main)

Let $TM = (\mathcal{A}, b, Q, q_0, \delta, Q_{accept}, Q_{reject})$ *be any single-tape Turing machine that has time complexity* $T(x)$ *and space complexity* $S(x)$ *on input* $x \in (\mathcal{A}\backslash\{b\})^*$. *There exists a transformer with constant depth, constant embedding dimension, Gated ReLU activation, and positional embedding* $n \mapsto n$, *average hard attention, such that for the next-token predictor* π_θ *implemented by this transformer and the reduction rule* ϕ' *defined in* (15), *the following holds:*

1. *PENCIL* $_{f_{\pi_\theta}, \phi'}$ *produces the same output (accept or reject) as TM on* x.
2. *The total number of tokens generated by* $\mathrm{PENCIL}_{f_{\pi_\theta}, \phi'}$ *is* $\mathcal{O}(T(x))$.
3. *The maximal context length used by* $\mathrm{PENCIL}_{f_{\pi_\theta}, \phi'}$ *during generation is at most* $\mathcal{O}(S(x))$.

Proof Scheme

We to construct a learnable model that can replicate PENCIL's model generation process, since the reduction process can be realized by the reduction rule. Specifically, at each iteration i, starting from a compressed state:

$$x^{(i-0.5)} \triangleq s \circ f_\pi^{t_i-1}(x) \in \Sigma^* \tag{19}$$

we need to construct a model that can autoregressively produce the extended sequence

$$x^{(i)} \triangleq \left(f_\pi^{t_i-t_{i-1}} \circ s \circ f_\pi^{t_i-1}(x), [\mathrm{SEP}], s \circ f_\pi^{t_i}(x), [\mathrm{RETURN}]\right) \in \Sigma^* \tag{20}$$

Proof Scheme

The base case $x^{(0.5)} \triangleq x$ serves as the initial prompt. Iteration i then starts from $x^{(i-0.5)}$ and ends with $x^{(i)}$.

Here

- $\pi : \hat{\Sigma}^* \to \hat{\Sigma}$ is the next-token generator in the autoregressive machine that simulates Turing Machine, where $\hat{\Sigma} = \mathcal{Q} \times \mathcal{A} \times \{-1, 0, 1\}$.

To implement this mapping, PENCIL uses a transformer as the next-token generator $\pi_\theta : \Sigma^* \to \Sigma$ where transformer vocabulary is $\Sigma \triangleq \hat{\Sigma} \cup \{[\mathrm{SEP}], [\mathrm{RETURN}]\}$ and θ is the transformer parameter.

It suffices to show that there is a next-token generator $\pi' \in \mathrm{FASP}\,[n; [\cdot]_+, \times]$(or equivalently, expressible by a transformer with $n \mapsto n$ positional embedding, average-hard attention and Gated ReLU activation) that can

1. simulate the next-token generator in the autoregressive machine that simulates Turing Machine.
2. generate the special token [SEP] at the earliest time that the length will be halved after summarization.
3. simulate the summarization process.

Transformer Construction as FASP Program

```
# Detect separator token
is_sep = (get_token = onehot([SEP]))
exist_sep = seq_or(is_sep)

# Phase masks to distinguish between simulation and summarization phases
sim_phase_mask = not exist_sep
sum_phase_mask = exist_sep and (not is_sep)

# Position tracking for Simulation, frozen in SUMMARIZATION (after [SEP] is generated)
next_sim_pos = seq_sum(get_move and sim_phase_mask)
current_sim_pos = next_sim_pos - (get_move and sim_phase_mask)
max_pos = seq_max(current_sim_pos)
min_pos = seq_min(current_sim_pos)
expected_sum_len = max_pos - min_pos + ReLU(max_pos- next_sim_pos -1) + 1

# SIMULATION Phase
# Get current symbol at head position
current_symbol = rightmost_exact_match(next_sim_pos,current_sim_pos,get_symbol,onehot(b))
# Compute next step based on transition function
simulation_step = transition(get_state, current_symbol)

# Decide whether to continue simulation or switch to summarization
end_simulation = sequence_len >= 2* expected_sum_len
simulation=if_then_else(end_simulation, onehot([SEP]), simulation_step)

# SUMMARIZATION Phase
current_sum_pos = seq_sum(get_move and sum_phase_mask)
current_sum_len = seq_sum(sum_phase_mask)

# Decide the next move in SUMMARIZATION PHASE
next_move = compute_move(current_sum_len, next_sim_pos, max_pos, min_pos)

# By construction, exact match always happens.
summary_symbol=rightmost_best_match(current_sum_pos+min_pos,current_sim_pos,get_symbol)
summary_step = get_state ⊗ summary_symbol ⊗ onehot(next_move)

# Check if we've reached the final position in summarization
end_summary = (current_sum_len = expected_sum_len)
summary = if_then_else(end_summary, onehot([RETURN], summary_step))

# MAIN - Select appropriate action based on current phase
result = if_then_else(exist_sep, summary, simulation)
```

Content

Summary and Take Away

- PENCIL adopts **a simple reduction rule** to "clean up" unneeded reasoning steps as soon as they are finalized, therefore enabling efficient training and allowing the model to handle substantially larger problems under the same memory constraints.
- PENCIL using transformers as the base model can simulate Turing machines with optimal efficiency in both time and space.

Acknowledgment

- Thanks Dr. Tianhao for the comprehensive instruction of the mind flow for this presentation.
- Thanks Zhi, Binghua, Mengzhe for all the great thoughts and inspiring questions.

References I


Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2006).
Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t).
Journal of the ACM (JACM), 53(6):937–977.

Content

Bounding the Maximum Sequence Length (Space)

Consider any point immediately before the [RETURN] of iteration i. By definition of t_i, we have

$$\left|s \circ f_\pi^{t_i-1}(x)\right| > \frac{1}{2}\left|f_\pi^{t_i-1-t_{i-1}} \circ s \circ f_\pi^{t_{i-1}}(x)\right|$$

Hence, if we look at the entire sequence (87) its length is at most

$$2\left|s \circ f_\pi^{t_i-1}(x)\right| + 2 + \left|s \circ f_\pi^{t_i}(x)\right| + 2 = \mathcal{O}(S(\mathcal{M}, s, x))$$

Here the additional " $+2$ " accounts for the two special tokens [SEP] and [RETURN], plus a small constant overhead. Because $s \circ f_\pi^{t_i-1}(x)$ (and also $s \circ f_\pi^{t_i}(x)$) is at most $S(\mathcal{M}, s, x)$ in length, we conclude that at every [RETURN], the sequence is $\mathcal{O}(S(\mathcal{M}, s, x))$ long. This implies the maximum context length under PENCIL never exceeds $\mathcal{O}(S(\mathcal{M}, s, x))$.

Bounding the Total Number of Tokens (Time)

Next, we show the total tokens generated (summing over all iterations) is $\mathcal{O}(T(\mathcal{M}, x))$. The critical point is that our reduction rule does not trigger too frequently: if we were to compress immediately after every single token (e.g. each Turingmachine step), we would incur an excessive time overhead. By only reducing when the sequence grows sufficiently large relative to the state size, we avoid inflating the total time cost. Formally, define

$$\ell_i \triangleq (t_i - t_{i-1}) + \left|s \circ f_\pi^{t_i}(x)\right| + 2$$

which represents the cost (length) of generating the new tokens in iteration i, plus the two special tokens (such as [SEP] and [RETURN]).

Bounding the Total Number of Tokens (Time)

We wish to bound $\sum_{i=1}^{I} \ell_i$. From the definition of t_i, it follows that

$$(t_i - t_{i-1}) + \left|s \circ f_\pi^{t_i}(x)\right| \geq 2\left|s \circ f_\pi^{t_{i-1}}(x)\right|$$

Summing up (92) from $i = 1$ to I gives us

$$(t_I - t_0) + \left|s \circ f_\pi^{t_I}(x)\right| \geq \sum_{i=1}^{I} \left|s \circ f_\pi^{t_{i-1}}(x)\right|$$

where $\left|s \circ f_\pi^{t_0}(x)\right| = 0$. Since $t_I \leq T(\mathcal{M}, x)$ (the total number of steps for $\mathcal{M}$), each iteration's generation cost can be bounded by a linear function of t_I plus the space used by the states. Concretely, summing up ℓ_i over i yields

$$\sum_{i=1}^{I} \ell_i \leq \sum_{i=1}^{I} \left[(t_i - t_{i-1}) + \left|s \circ f_\pi^{t_i}(x)\right| + 2\right] \leq 2t_I + 2I + \left|s \circ f_\pi^{t_I}(x)\right|$$

Bounding the Total Number of Tokens (Time)

Since $I \leq t_I$ (each iteration covers at least one time step) and $t_I \leq T(\mathcal{M}, x)$, we conclude $\sum_{i=1}^{I} \ell_i = \mathcal{O}(T(\mathcal{M}, x))$

Some Operators

- *transition*:(state, symbol) $\mapsto$ (next state,next symbol, next move)
- *seq_or*:$seq\_or(\psi)(x) = \bigvee_{j=1}^{n} \psi(x_{1:j})$
- *rightmost_exact_match*:

Rightmost Exact Match For any positive integer d, d', we define $\texttt{rightmost\_exact\_match} : \mathcal{H}(\mathbb{Z}^{d'}) \times \mathcal{H}(\mathbb{Z}^{d'}) \times \mathcal{H}(\mathbb{R}^{d}) \times \mathcal{H}(\mathbb{R}^{d}) \to \mathcal{H}(\mathbb{R}^{d})$ as the variant of rightmost best match (and thus variant of rightmost hard attention) which returns the value ψ_v associated with the rightmost key ψ_k that exactly matches the query ψ_q, and otherwise returns the default value ψ_d. That is, for any $x \in \Sigma^n$:

$$\begin{aligned} &\texttt{rightmost\_exact\_match}(\psi_q, \psi_k, \psi_v, \psi_d)(x) \\ \triangleq &\texttt{if\_then\_else}(\texttt{rightmost\_best\_match}(\psi_q, \psi_k, \psi_k) = \psi_q, \psi_v, \psi_d). \end{aligned} \tag{80}$$