

Chenxiao Yang

PENCIL: Long Thoughts with short Memory

@FLann May 26th, 2025







Nathan Srebro







David McAllester









Chain-of-Thought (CoT)



StrategyQA

Q: Yes or no: Would a pear sink

A: The density of a pear is about 0.6 g/cm³, which is less than water. Thus, a pear would float.

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



More Test-Time Computes

Performance of O3 on ARC-AGI-Pub

tasks (Feng et al. 2024, Merill et al. 2024, Li et al. 2024, etc.)



Niklas Muennighoff^{*134} Zitong Yang^{*1} Weijia Shi^{*23} Xiang Lisa Li^{*1} Li Fei-Fei¹ Hannaneh Hajishirzi²³ Luke Zettlemoyer² Percy Liang¹ Emmanuel Candès¹ Tatsunori Hashimoto¹



S1, Muennighoff et al., 2025

Long CoT empowers transformers to solve computationally more intensive



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. onger 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. longer usefu

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*.

 \rightarrow **Final answer**: your name.





"Erasure" is Fundamental to Computation



Turing Machine (1936)

Modern Computer

One Step of Thinking \Leftrightarrow **One Step of Computation**







Transformer

ectrone ? at pure a Calbrilicon area **PENCIL ENables Context**efficient Inference and Learning

Printer Bashrand

idespaceto.

Generalden

antippe

a very signation

Conserve Sugar bar and and an ward and

presidente and a far the second and a

he a shart shart and the

and george as and the

An Low some how and

art spineter Corrie WHO PRODUCE STATES states a strategy and a state a grand approved a by J'ndow'

The facultar

and to find a course to ease los and a win frances binne tome fait habelone and the of one survey and have be a greater to far the for have been a farthe Carbon Ber and prove of a harmon for a same ber francia a har a far a ser a far a fa ator of anone contract and a state in a state of a state of the second and a state of the same a the same as a chiefles and and a fart are not for the floor to a dial

Courses de la s

uploged in secondaria

1 stream or no 200

-262 2 10 25777

more manufactor

Dig inserver

allepton als metricali

Barrow pro Bessiness dessentations and and as a to any starte a say popular Coffeetres de manerfrees ca and Charles and Alere bet fan e

found mentals. 09/01-11:00 restance and inada and reasons dreps in the of a to be a to be a to a to a to a to -vertine giveranaloradorador alo 是477年3月 where responde Lechertra and 10000000000 when ore washes WHERE MEY PRESERVED and Spridge St. 4 CETTERSE 2007 Jesperandosks.

Alas a character and KEPPETRA HONR and water active that a the man

and our de de la ser a de la la faire de la de la de de de de de la de l They are a house and a set of the set of the set of the set of the a marga and the second of a second differences of the cal

- United and

Crebero Ann

Antolorena

Folewaren !!

te operation

rescent and

eleteror are

area gran

ansine of the

ALGNAHESS.

Car

superior presentations



acatra so



Model Generation (Write)



Autoregressive Next-token Generation





Reduction Rule (Erase)

	🔓 examples.lisp
;; Common Lisp examples.	; SLIME 2.26 CL-USER> (bello-world)
<pre>(defun hello-world () "Print 'hello, world' message." (format t "hello, world~%"))</pre>	hello, world NIL CL-USER> (factorial 6)
<pre>(defun factorial (n) "Compute factorial of n." (if (= n 0) 1 (* n (factorial (- n 1)))))</pre>	CL-USER> (trace factorial) (FACTORIAL) CL-USER> (factorial 6) 0: (FACTORIAL 6) 1: (FACTORIAL 5)
<pre>defun fibonacci (n) "Compute nth Fibonacci number." (if (< n 2) n (+ (fibonacci (- n 1)) (fibonacci (- n 2))))</pre>	2: (FACTORIAL 4) 3: (FACTORIAL 3) 4: (FACTORIAL 2) 5: (FACTORIAL 1) 6: (FACTORIAL 0) 6: FACTORIAL returned 1 5: FACTORIAL returned 1
	4: FACTORIAL returned 2 3: FACTORIAL returned 6 2: FACTORIAL returned 24 1: FACTORIAL returned 120 0: FACTORIAL returned 720 720
	CL-USER> (fibonacci 6) 8 CL-USER>
-: examples.lisp All L17 (Lisp adoc [COMMON-	-LISP-USIU:**- *slime-repl sbcl* All L23 (REPL adoc

Functional Programming





Model Generation (Write) Reduction Rule (Erase)



C [CALL] T [SEP] A [RETURN] \Rightarrow CAContext $\in \Sigma^*$ Thoughts $\in \Sigma^*$ Answer $\in \Sigma^*$ Special Tokens

- Reduction is triggered when the sequence matches the pattern
- PENCIL iteratively generate thoughts and *friggers* 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.afferoywEwerestoy phitabdfictbagagshowbwoght-wwekbowdbbagogmobnynpwdhodahafyndoyhe5 ere invallhogdcbygaddmogkipgybbgBfroddbbagsafferoddbagsafferoddbbagsafferod

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

This is a hypothetical example for illustration purpose

11

Example 1: Arithmetic Expression Evaluation

used in total?



5 toys each and 2 red bags with 4 toys each. How many toys were used in total?

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

A toy store put together party bags for a birthday. They made 3 blue bags with



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 $3 \times 5 = 15$. 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 $2 \times 4 = 8$. 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.





	♥
	[EndOfPrompt] [CALL] Let's break this problem down into parts! [CALL] 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 $3 \times 5 = 15$. [SEP] There were 15 toys in all
	blue bags. [RETURN]
	♥
:([EndOfPrompt] [CALL] Let's break this problem down into parts! There were 15 toys in all blue bags.
	\checkmark
([EndOfPrompt] [CALL] Let's break this problem down into parts! There were 15 toys in
	all blue bags. [CALL] 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. [SEP] There were 8 toys in all red bags. [RETURN]
	♥
([EndOfPrompt] [CALL] Let's break this problem down into parts! There were 15 toys in
	all blue bags. There were 8 toys in all red bags.
	♥
([EndOfPrompt] [CALL] Let's break this problem down into parts! There were 15 toys in
	all blue bags. 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: $15 + 8 = 23$.
	[SEP] There were 23 toys used in total. [RETURN]
	♥
([EndOfPrompt] There were 23 toys used in total .
([EndOfPrompt] There were 23 toys used in total. [EndOfText]



Example 2: Quantified Boolean Formula (QBF)

 $\exists X_2 \forall X_1: (X_2 \vee \neg X_2 \vee X_1) \land (X_1 \vee X_2) \land (X_2) \land (\neg X_2 \vee \neg X_1) \land (X_1 \vee \neg X_1) \land (\neg X_1 \vee \neg X_2) \land fswer] i \$r \forall alse X_2 = False \land fsyF \land fsyF$





PENCIL: length \propto poly(*n*)

14

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



PENCIL significantly reduces the maximal context length during inference $(\times 0.004 \text{ when } n=10 \text{ on } QBF)$





15

Thought Cought Thought Though Thought Thought



Experimental Setting: Training and Inference

Inference:

* Datasets are generated by running specialized algorithms

- **Training:** Key difference between CoT and PENCIL \Rightarrow **Data Generation***
 - $\mathscr{L}_{CoT} = -\sum \log p(\text{next token} | \text{complete sequence})$
 - $\mathscr{L}_{\text{PENCIL}} = -\sum_{n=1}^{\infty} \log p(\text{next token} | \text{reduced sequence})$
- We train a small transformer (25M parameter, 2048 context length) from scratch.

- C [CALL] T [SEP] A [RETURN] \Rightarrow C A
- Preserve the KV cache of **Context (C)** and recompute that for **Answer (A)**





Performance Comparison on 3-SAT and QBF

n =	3	4	5	6	7	8	9	10	n =	3	4	5	6	7	8	9	10
Baseline	66	57	46	51	46	51	49	51	Baseline	90	82	85	68	60	69	71	66
СоТ	100	100	100	99	84	63	54	50	СоТ	100	100	97	94	74	72	69	73
PENCIL	100	100	100	99	99	100	100	100	PENCIL	100	100	100	100	100	100	100	100

3-SAT

QBF

NENCIL 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

NOTE NOTE CONTRUE CON





QBF n=4

NOT. **PENCIL** is computationally more efficient than **CoT**.

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

QBF n=5

QBF n=6



21

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?

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

Solution :





Example 3: Einstein's Puzzle



(b) Summarize state changes and update possibilities.

Special Usage: Summarization

Long Thoughts Summarized Thoughts

python

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

print(factorial_tail(5)) $\# \rightarrow 120$

The "returned value" is another "function call" \Rightarrow A (Answer) = [CALL] T'

- C [CALL] T [SEP] [CALL] T' [RETURN] \Rightarrow C [CALL] T'
 - **Tail recursion** in functional programming:

Copy



Performance on Einstein's Puzzle

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

NOTE NOTE 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

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

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

QBF

Einstein's Puzzle





How Powerful is

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 = 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 NENCIL

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.







Step : simulating a computation step of TM

: the current configuration of TM (written symbols) State

Strategy: Iterative "Think" and "Summarize"



When?	Never (
# Tokens Generated	O(Time) 🔽
Max Context Length	O(Time) 🔀

Recall we use [CALL] T [SEP] [CALL] T' [RETURN] \Rightarrow [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

<pre>Detect separator token s_sep = (get_token = onehot([SEP])) This is the Proof !</pre>
exist_sep_=_seq_or(is_sep) entitle limitation or switch to summarization end_simulativallalalalaeeee expected_sum_len Phase_masks_to_distinguish_between_simulation_and_summarization_phases_step)
im_phase_mask = not exist_sep
um_pnase_mask = exist_sep and (not is_sep)
<pre>current_sum_pos = seq_sum(get_move and sum_phase_mask) Position_tracking_for_Simulation, frozen_in_SUMMARIZATION (after [SEP] is generated) ext_sim_pos = seq_sum(get_move and sim_phase_mask)</pre>
current_sim_pos = next_sim_pos - (get_move and sim_phase_mask)
ax_pos_seq_max(current_sim_pos/rrent_sum_len, next_size_pol, need, of pol pol pol pol
<pre>in_pos = seq_min(current_sim_pos)</pre>
expected_sum_len = max_pos - min_pos + ReLU(max_pos - next_sim_pos -1) + 1
<pre>summary_symbol=rightmost_best_match(current_sum_pos+min_pos,current_sim_pos,get_symbol SIMULATION_Phasep = get_state & summary_symbol & onehot(next_move) Get current symbol at head position</pre>
urrent_symbol = rightmost_exact_match(next_sim_pos,current_sim_pos,get_symbol,onehot(b))
Compute next step based on transition function sum len
<pre>imulation_step = transition(get_state, current_symbol)_RN], summary_step))</pre>
MAIN - Select appropriate action based on current phase

select appropriate action based on current phase result = if_then_else(exist_sep, summary, simulation)

returned variable





Future Directions & Open Questions Long Thoughts with Short Memory

more effectively teach them to reason in a structured way?

rules) that are even more efficient than PENCIL?

help guide practice?

• How to incorporate PENCIL into real-world LLM systems? How to

• Does there exist other "erasing" mechanisms (e.g. other reduction

• Are there any other perspectives from which theories in TCS can



Takeaways

- 3. Empirically, **YENCIL** enables longer and deeper thinking using shorter context, and thus can scale up to handle more complicated tasks.
- 4. Theoretically, **\Science PENCIL** is Turing-complete with optimal space and time complexity, and thus can solve arbitrary computable problems efficiently.

Thanks !

