Chain of Thought Prompting

STAT 9911
Yiwen Lu, Lu Li, PhD candidates in UPenn AMCS
April 17, 2025

From Chains to Graphs: Evolving Thought Structures in Large Language Models

Yiwen Lu
April 17, 2025

Why Chain-of-Thought Prompting?

- LLMs excel at language generation, but struggle with multi-step reasoning.
- Traditional prompting
  a. direct input-output mapping;
  b. lacks intermediate reasoning.
- Chain-of-Thought prompting elicits reasoning by prompting step-by-step solutions.
- Inspired by how humans explain logic (e.g., showing work in math problems).

Example of CoT

- CoT provides a natural language rationale.
- Easier to debug and understand model reasoning.
- The chain breaks down the problem, just like how we would solve it on paper

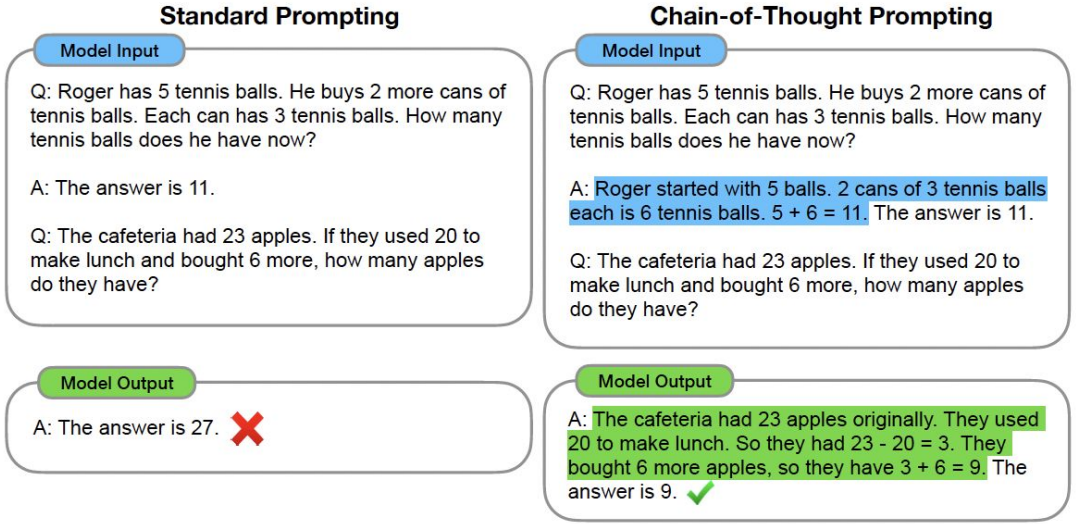


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

How CoT Prompting Works?

- Uses few-shot prompting with demonstrations of chain-of-thought.
- Input format: ⟨Question, CoT reasoning, Answer⟩
- No model finetuning is required — relies entirely on prompt engineering.
- Chains are sampled as single continuations during inference (greedy or stochastic decoding).
  - "Single continuation": treats the entire response — reasoning steps + final answer — as one long sequence of tokens to predict
  - Greedy decoding selects just one reasoning path — prone to:
    - Local optima,
    - Early mistakes propagating forward,
    - Lack of robustness.
  - Stochastic (Sampling-Based) Decoding
    - Introduces randomness to encourage diversity in reasoning paths.
    - Used in **CoT-SC (Self-Consistency)**: multiple reasoning paths are sampled, and the final answer is chosen by majority vote.

Overview of Self-Consistency in CoT

SELF-CONSISTENCY IMPROVES CHAIN OF THOUGHT REASONING IN LANGUAGE MODELS

Xuezhi Wang[†‡] **Jason Wei**[†] **Dale Schuurmans**[†] **Quoc Le**[†] **Ed H. Chi**[†]
Sharan Narang[†] **Aakanksha Chowdhery**[†] **Denny Zhou**[†§]
[†]Google Research, Brain Team
[‡]xuezhiw@google.com, [§]dennyzhou@google.com

- **Step 1:** Prompt with CoT – Include few-shot examples with step-by-step reasoning.
- **Step 2:** Sample Diverse Reasoning Paths – Use temperature/top-k sampling to generate many responses.
- **Step 3:** Aggregate Final Answers – Take majority vote or weighted average to pick the most consistent answer.

Overview of Self-Consistency in CoT

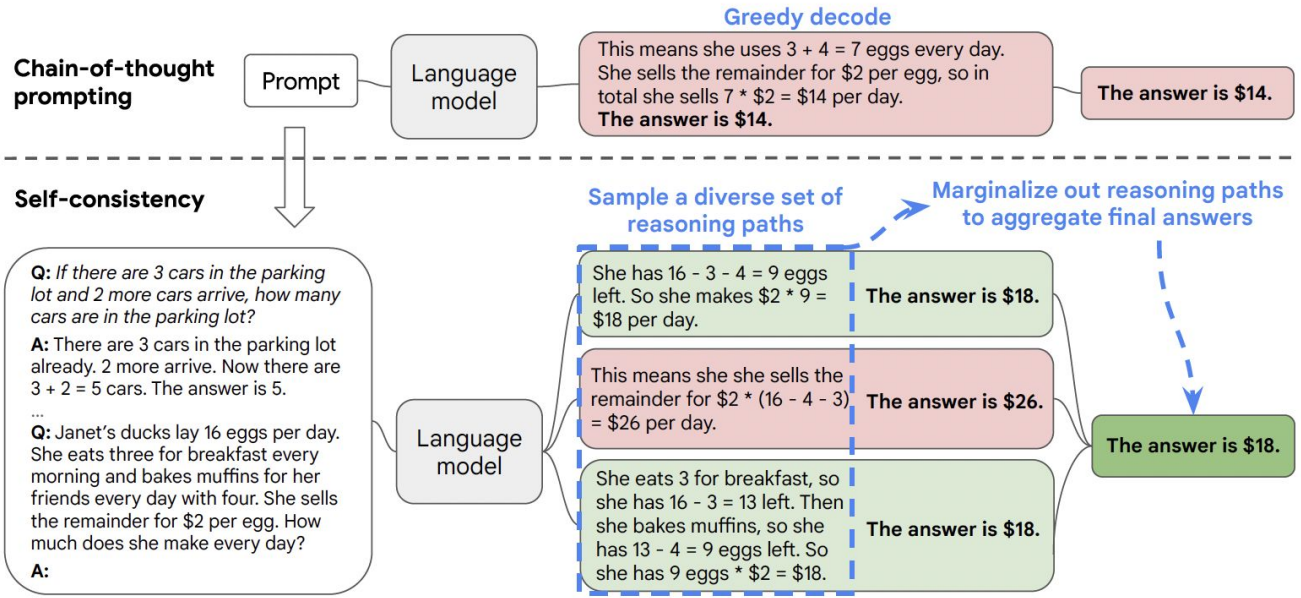


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the "greedy decode" in CoT prompting by sampling from the language model's decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set.

Why it works?

- Humans often solve problems in **different ways** and converge on correct answers.
- Incorrect reasoning paths are **less likely** to converge on the same result.
- Self-Consistency captures this by:
  - Sampling many plausible thoughts,
  - Filtering for consensus,
  - Providing a **“self-ensemble”** over one LLM.

SC introduces a latent variable

r_i = reasoning path,

Generates pairs (r_i, a_i)

Final answer a is chosen by:

$$\arg\max_a \sum_{i=1}^{m} \mathbf{1}(a_i = a)$$

Arithmetic Reasoning

- Compare CoT vs Self-Consistency

	Method	AddSub	MultiArith	ASDiv	AQuA	SVAMP	GSM8K
	Previous SoTA	**94.9**[a]	60.5[a]	75.3[b]	37.9[c]	57.4[d]	35[e] / 55[g]
UL2-20B	CoT-prompting	18.2	10.7	16.9	23.6	12.6	4.1
	Self-consistency	24.8 (+6.6)	15.0 (+4.3)	21.5 (+4.6)	26.9 (+3.3)	19.4 (+6.8)	7.3 (+3.2)
LaMDA-137B	CoT-prompting	52.9	51.8	49.0	17.7	38.9	17.1
	Self-consistency	63.5 (+10.6)	75.7 (+23.9)	58.2 (+9.2)	26.8 (+9.1)	53.3 (+14.4)	27.7 (+10.6)
PaLM-540B	CoT-prompting	91.9	94.7	74.0	35.8	79.0	56.5
	Self-consistency	93.7 (+1.8)	99.3 (+4.6)	81.9 (+7.9)	48.3 (+12.5)	86.6 (+7.6)	74.4 (+17.9)
GPT-3 Code-davinci-001	CoT-prompting	57.2	59.5	52.7	18.9	39.8	14.6
	Self-consistency	67.8 (+10.6)	82.7 (+23.2)	61.9 (+9.2)	25.6 (+6.7)	54.5 (+14.7)	23.4 (+8.8)
GPT-3 Code-davinci-002	CoT-prompting	89.4	96.2	80.1	39.8	75.8	60.1
	Self-consistency	91.6 (+2.2)	**100.0** (+3.8)	**87.8** (+7.6)	**52.0** (+12.2)	**86.8** (+11.0)	**78.0** (+17.9)

Commonsense and Symbolic Reasoning

- Yes/no implicit reasoning

	Method	CSQA	StrategyQA	ARC-e	ARC-c	Letter (4)	Coinflip (4)
	Previous SoTA	**91.2**[a]	73.9[b]	86.4[c]	75.0[c]	N/A	N/A
UL2-20B	CoT-prompting	51.4	53.3	61.6	42.9	0.0	50.4
	Self-consistency	55.7 (+4.3)	54.9 (+1.6)	69.8 (+8.2)	49.5 (+6.8)	0.0 (+0.0)	50.5 (+0.1)
LaMDA-137B	CoT-prompting	57.9	65.4	75.3	55.1	8.2	72.4
	Self-consistency	63.1 (+5.2)	67.8 (+2.4)	79.3 (+4.0)	59.8 (+4.7)	8.2 (+0.0)	73.5 (+1.1)
PaLM-540B	CoT-prompting	79.0	75.3	95.3	85.2	65.8	88.2
	Self-consistency	80.7 (+1.7)	**81.6** (+6.3)	**96.4** (+1.1)	**88.7** (+3.5)	70.8 (+5.0)	91.2 (+3.0)
GPT-3 Code-davinci-001	CoT-prompting	46.6	56.7	63.1	43.1	7.8	71.4
	Self-consistency	54.9 (+8.3)	61.7 (+5.0)	72.1 (+9.0)	53.7 (+10.6)	10.0 (+2.2)	75.9 (+4.5)
GPT-3 Code-davinci-002	CoT-prompting	79.0	73.4	94.0	83.6	70.4	99.0
	Self-consistency	81.5 (+2.5)	79.8 (+6.4)	96.0 (+2.0)	87.5 (+3.9)	**73.4** (+3.0)	**99.5** (+0.5)

Repairing CoT Errors — Qualitative Examples

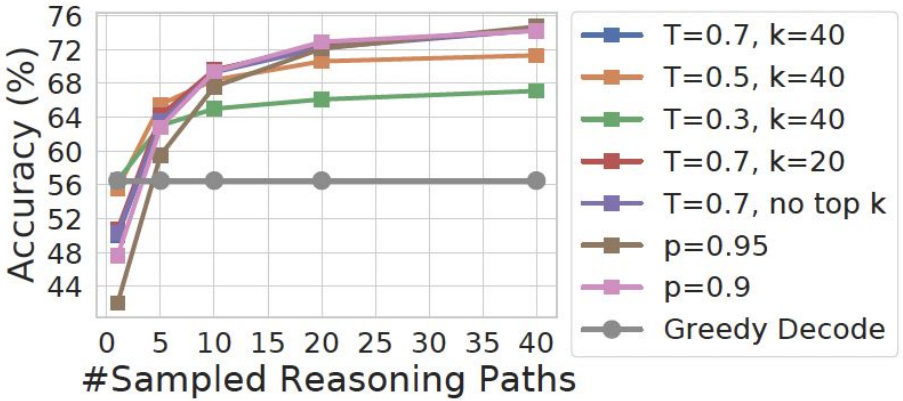
- SC fixes CoT reasoning mistakes

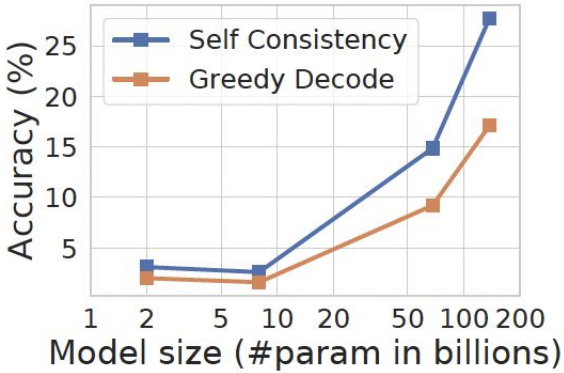
	GSM8K	StrategyQA
Question	*Henry made two stops during his 60-mile bike trip. He first stopped after 20 miles. His second stop was 15 miles before the end of the trip. How many miles did he travel between his first and second stops?*	*Yes or no: Is Albany, Georgia the most populous US Albany?*
Greedy Decode	He traveled 60 miles in total. He stopped after 20 miles, so he traveled 60 - 20 = 40 miles between the first and second stops. The answer is 40.	Albany, Georgia is the most populous US Albany. Thus, the answer is yes.
Sampled Path 1	Henry travelled a total of 60 miles. His first stop was 20 miles in, and his second stop was 15 miles before the end. So between his first and second stops he travelled 60 - 20 -15 = 25 miles. The answer is 25.	The most populous US Albany is Albany, New York. Thus, Albany, Georgia is not the most populous US Albany. So the answer is no.
Sampled Path 2	He made two stops during a 60-mile trip. The first was 20 miles into the trip. The second was 15 miles before the end of the trip. This means the second stop was 60 - 15 = 45 miles into the trip. Since he made the stops in order, the second stop must have been 45 - 20 = 25 miles after the first stop. The answer is 25.	Albany, Georgia has a population of about 88,000. Albany, New York has a population of about 95,000. Thus, Albany, Georgia is not the most populous US Albany. So the answer is no.

Table 4: Examples where self-consistency helps repair the errors over greedy decode, on PaLM-540B. Two sampled reasoning paths that are consistent with the ground truth are shown.

Robustness to Sampling Strategies and Model Scaling

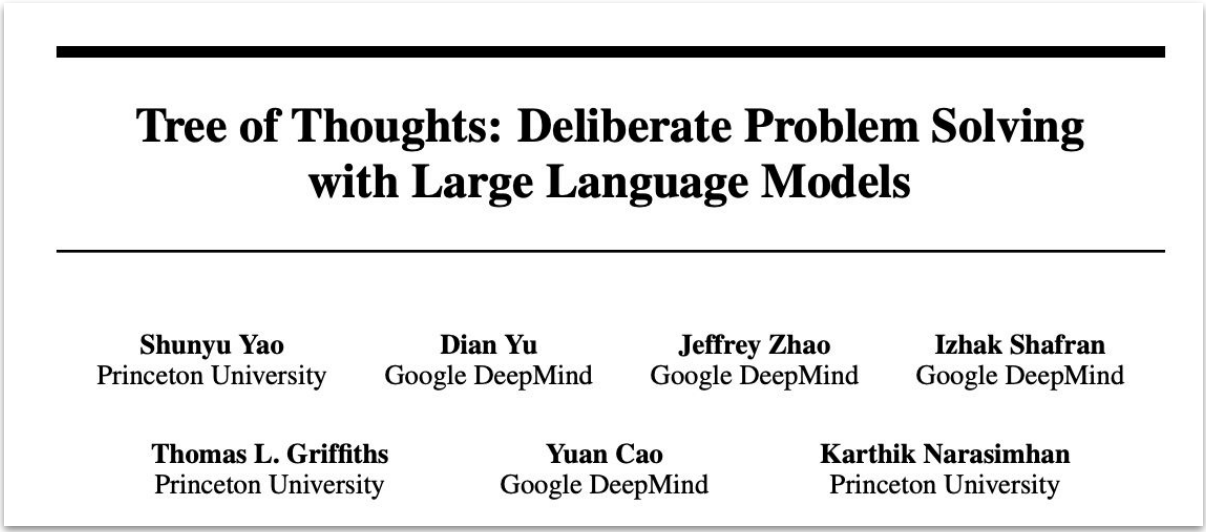
- Robust to Sampling Variants
  - Temperature settings (e.g., T = 0.3, 0.5, 0.7)
  - Top-k sampling (k = 20, 40)
  - Nucleus sampling (p = 0.9, 0.95)
  - Accuracy remains stable even under different combinations.
- Scales with Model Size
  - Performance improves monotonically with larger models.
  - Accuracy gain amplifies at higher scale (e.g., GPT-3 → PaLM-540B).
  - Suggests SC benefits from emergent reasoning ability in large models.





From Chain to Tree: Why We Need ToT ?

- LLMs typically generate outputs in a left-to-right sequence (CoT).
- Limitation: No ability to look ahead, backtrack, or reconsider.
- Inspired by human planning: Search through a problem space tree.
- Idea: Turn inference into deliberate tree search over thoughts.



Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Shunyu Yao Princeton University

Dian Yu Google DeepMind

Jeffrey Zhao Google DeepMind

Izhak Shafran Google DeepMind

Thomas L. Griffiths Princeton University

Yuan Cao Google DeepMind

Karthik Narasimhan Princeton University

ToT Framework

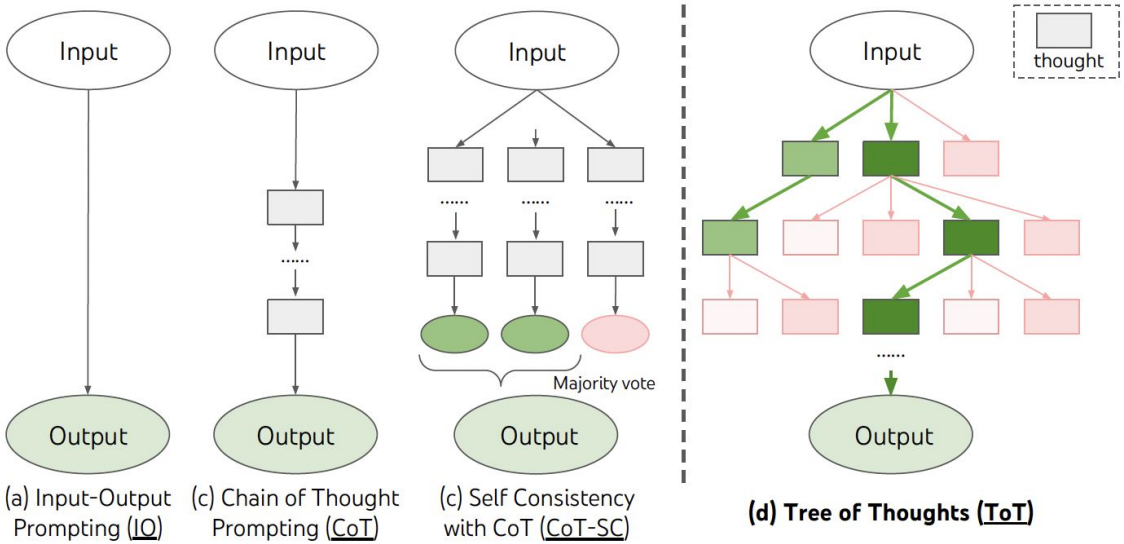


Figure 1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a *thought*, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 2, 4, 6.

ToT Framework

- **Thought Decomposition** — Break the problem into discrete steps.
- **Thought Generation** — Generate multiple candidates per step.
- **State Evaluation** — Score or vote on partial solutions.
- **Tree Search Algorithm** — Explore promising branches (BFS or DFS).

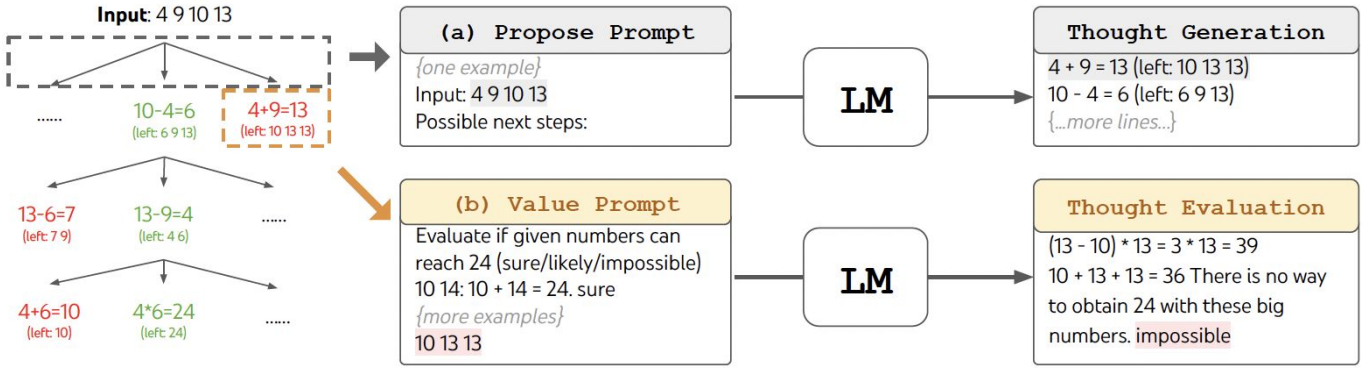
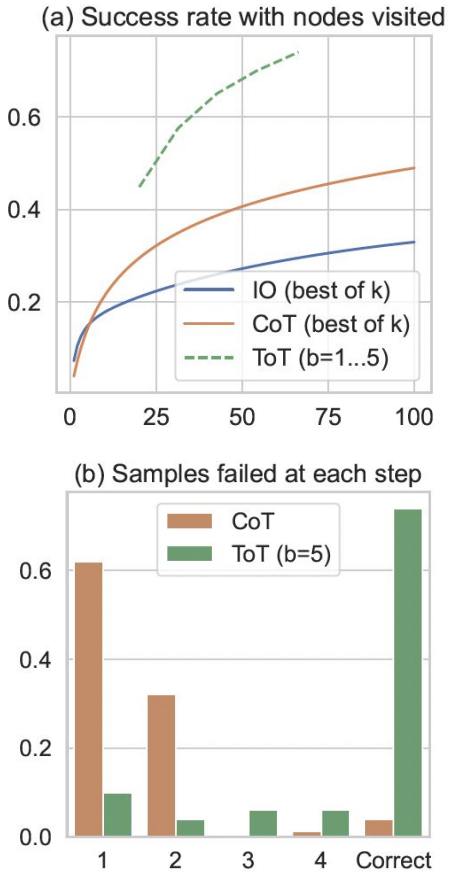


Figure 2: ToT in a game of 24. The LM is prompted for (a) thought generation and (b) valuation.

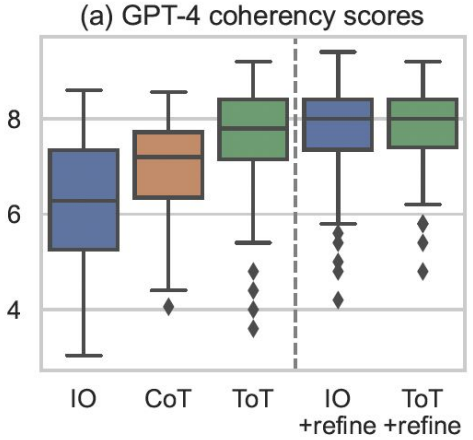
Applications - Game of 24 (math)

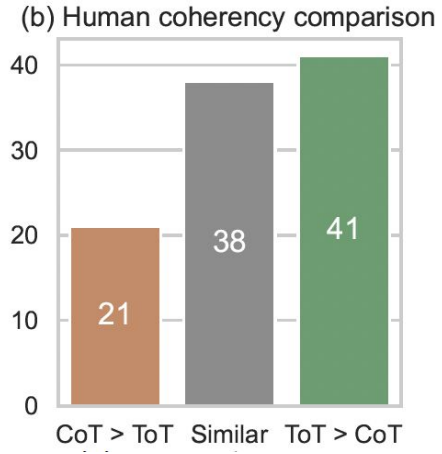
- Setup:
  - Input: 4 numbers, goal = 24
  - Output: Valid equation using each number once
  - Evaluation = success rate over 100 hard games
- Insight:
  - ToT enables early pruning and lookahead
  - CoT often fails on first equation step



Applications - Creative writing

- Setup:
  - Input: 4 random sentence constraints
  - Output: Passage of 4 paragraphs ending with those sentences
  - Evaluation:
    - GPT-4 zero-shot coherence score (1–10)
    - Human pairwise preference
- Insight:
  - ToT outperforms CoT in open-ended creative planning
  - Voting over plans and drafts improves coherence



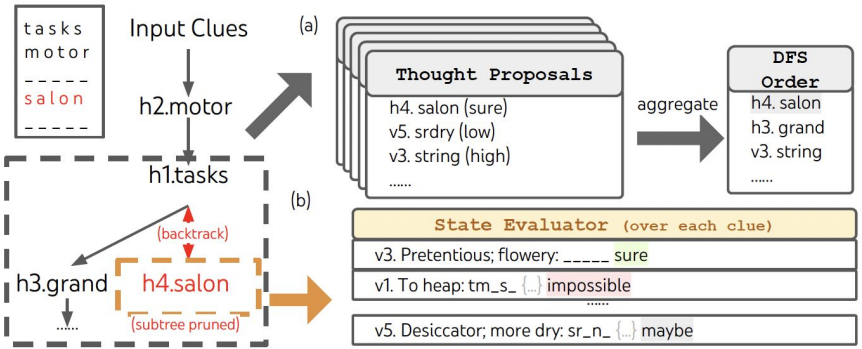


Applications - Crossword puzzles

- Setup:
  - Input: 10 clues (e.g., "Presented" = SHOWN)
  - Output: Valid 5×5 crossword grid
  - Evaluation: % of valid solutions (rows & columns = real words)
- Insight:
  - ToT uses depth-first tree search with pruning
  - Value prompt removes infeasible guesses early
  - CoT fails due to lack of planning over intersecting clues

Method	Success Rate (%) Letter	Word	Game
IO	38.7	14	0
CoT	40.6	15.6	1
ToT (ours)	**78**	**60**	**20**
+best state	82.4	67.5	35
-prune	65.4	41.5	5
-backtrack	54.6	20	5

Table 3: Mini Crosswords results.



Discussion

Strengths:

- Generalizable: Works across math, writing, lexical tasks
- Modular: Plug in different LMs, prompts, search methods
- Strategic: Supports lookahead, backtracking, voting
- No finetuning needed

Limitations:

- Cost: Requires multiple generations + evaluations
- Manual task-specific decomposition still required

Limitations of Trees → Graphs-of-Thoughts (GoT)

- CoT → linear reasoning
- ToT → structured but rigid branching
- GoT generalizes both – Vertices = thoughts, Edges = dependencies
- Supports aggregation, refinement, backtracking, and merging

Graph of Thoughts: Solving Elaborate Problems with Large Language Models

Maciej Besta[*1], Nils Blach[*1], Ales Kubicek[1], Robert Gerstenberger[1], Michał Podstawski[2], Lukas Gianinazzi[1], Joanna Gajda[3], Tomasz Lehmann[3], Hubert Niewiadomski[3], Piotr Nyczyk[3], Torsten Hoefler[1]

[1]ETH Zurich
[2]Warsaw University of Technology
[3]Cledar
bestam@inf.ethz.ch, nils.blach@inf.ethz.ch, htor@inf.ethz.ch

GoT Framework

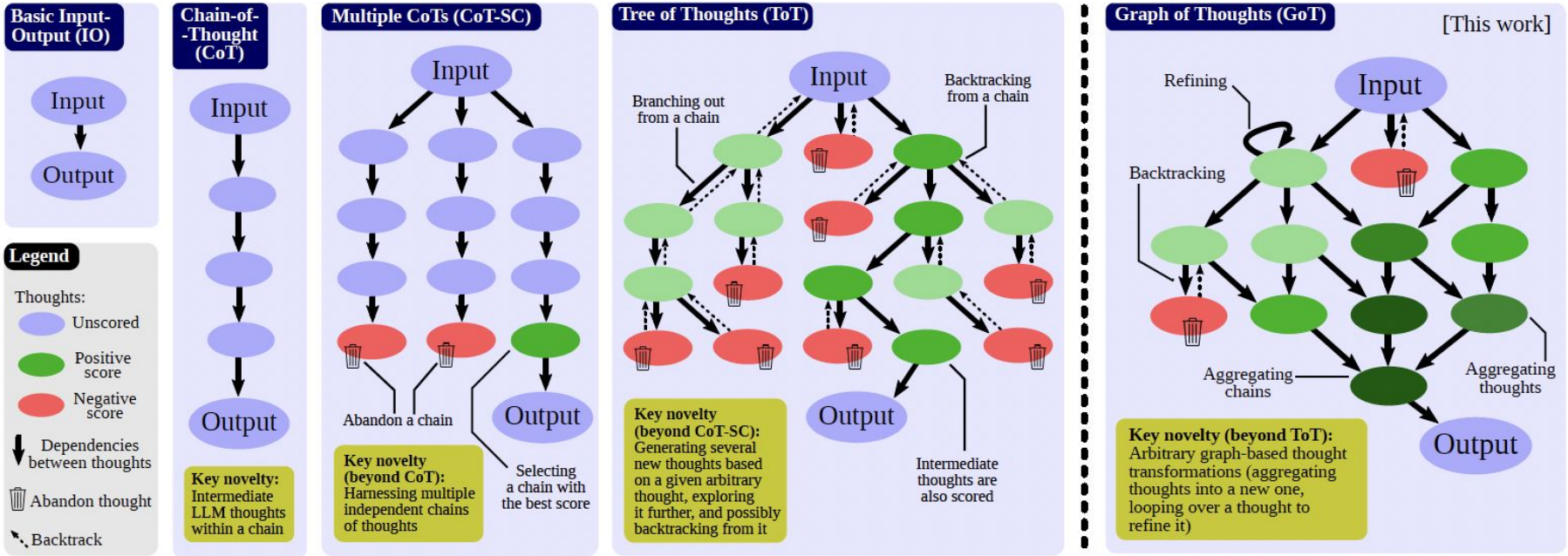


Figure 1: Comparison of Graph of Thoughts (GoT) to other prompting strategies.

GoT Framework

Key Concepts:

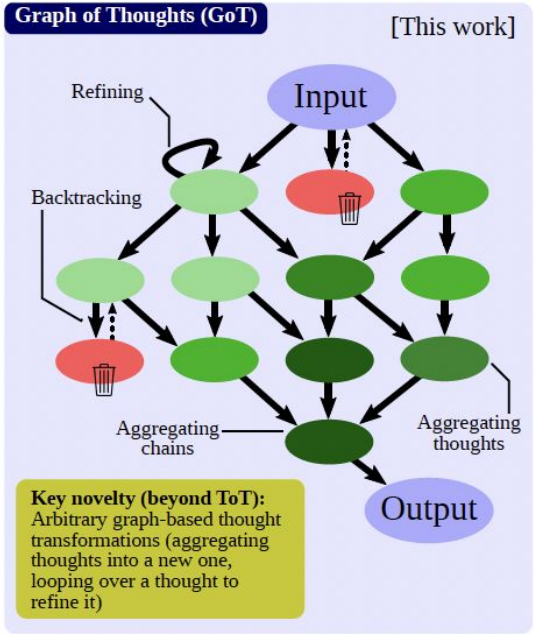
- Reasoning = Graph $G = (V, E)$
- Each vertex $v \in V$: a thought
- Each edge (v_i, v_j): dependency

- Graph enables transformations:

```
Aggregate(t1, ..., tn)
```

```
Refine(t)
```

```
Generate(t)
```



Modular System Architecture

Modules:

- Prompter: Builds LLM input
- Parser: Extracts thought state
- Scorer & Validator: Grades thoughts
- Controller: Executes plan
- GRS: Tracks reasoning state over time

Key Transformations:

- Aggregate: Merge multiple thoughts
- Loop (Refine): Re-enter the same thought
- Prune: Drop unhelpful vertices
- Branch: Explore new thoughts from any node

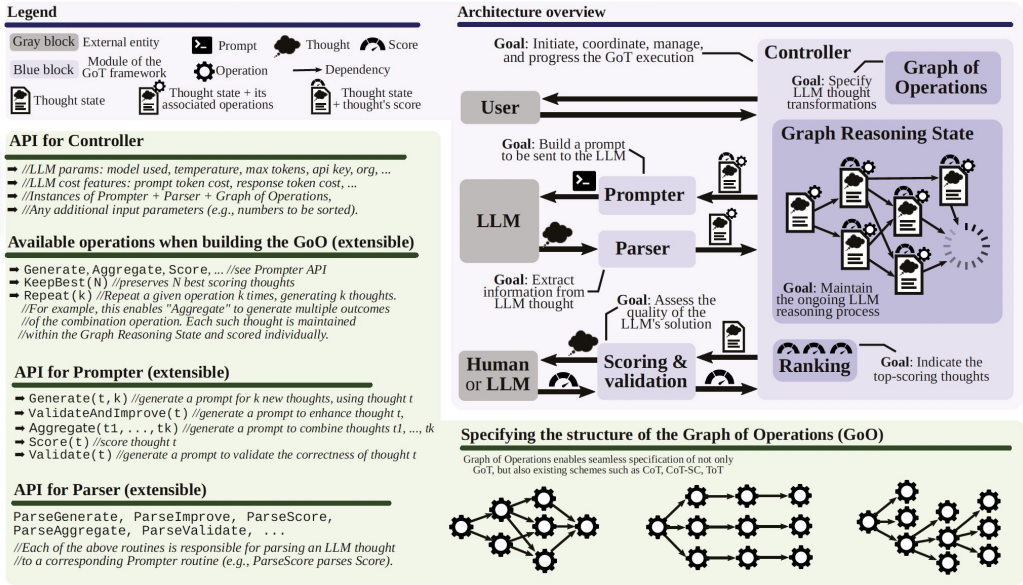
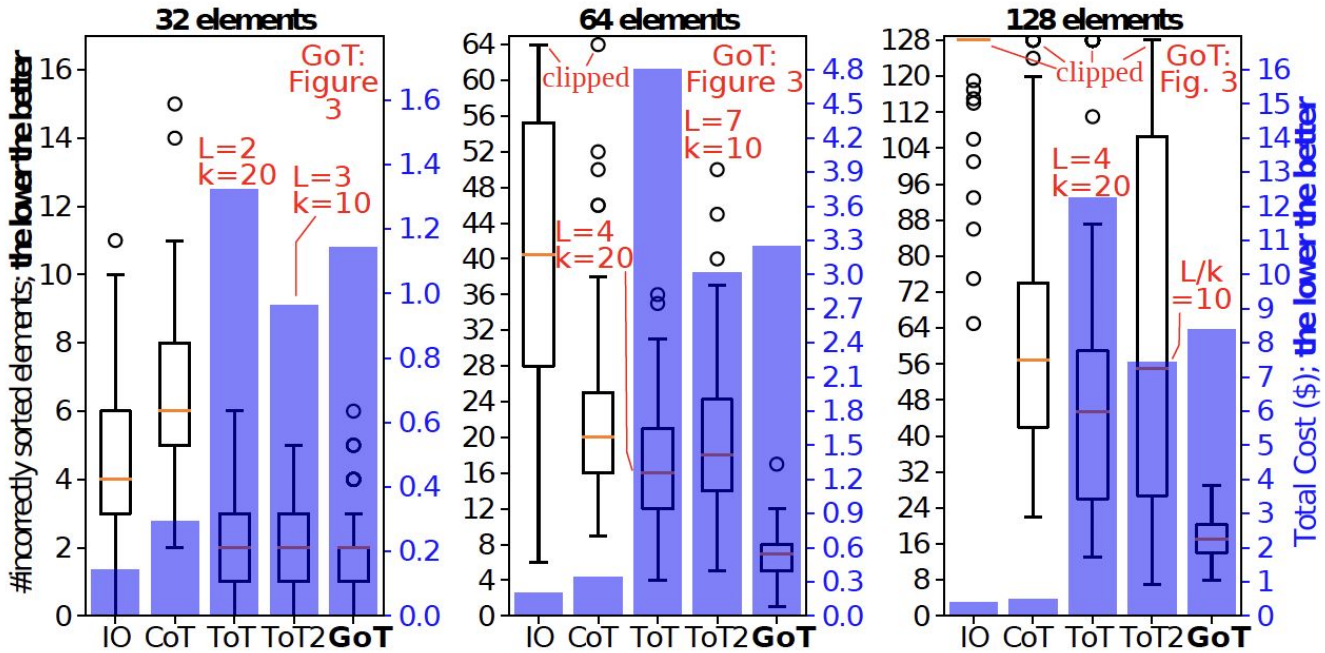


Figure 2: The system architecture of GoT, and the APIs of respective modules. The user can straightforwardly extend the design towards new prompting schemes, experiment with novel thought transformations, and plug in different LLMs. The blue part of the figure contains the architecture overview, and the green part lists the API.

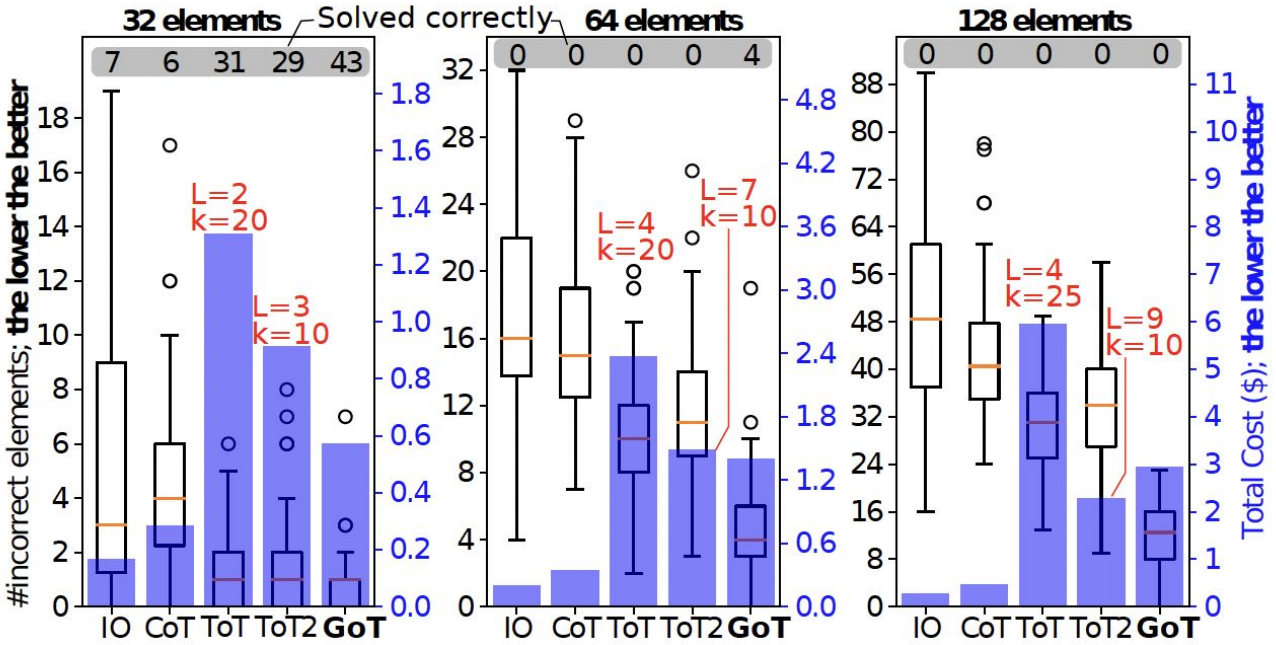
Use Cases and Evaluation - Sorting

Sort arrays of 32, 64, 128 elements



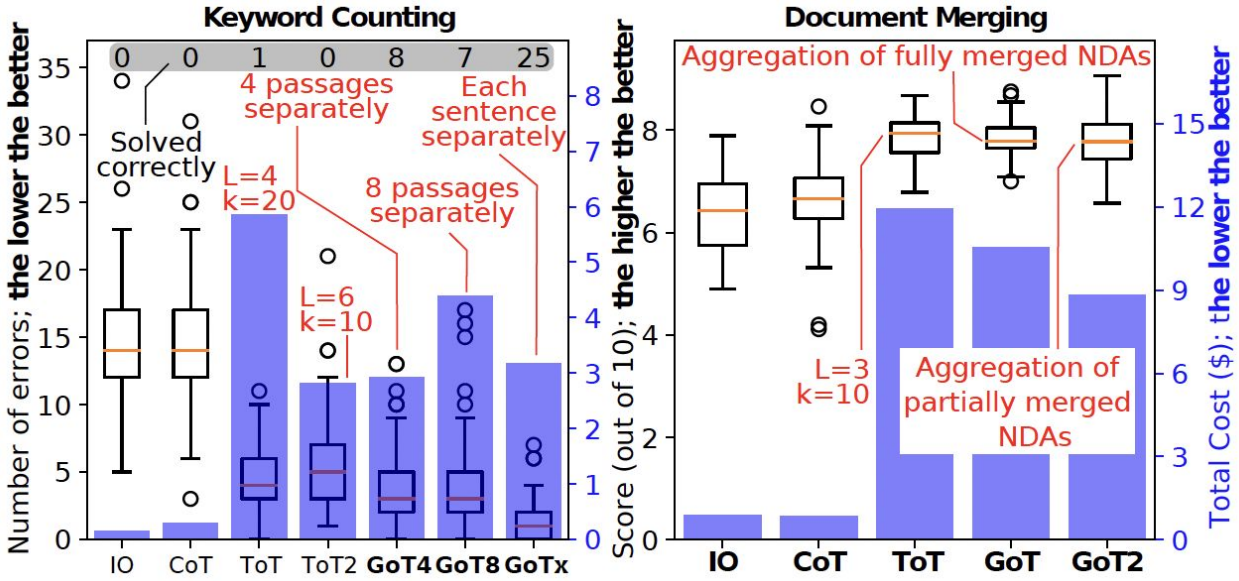
Use Cases and Evaluation - Set Intersection

Compute intersection of two sets (32, 64, 128 elements)



Use Cases and Evaluation - Keyword Counting and Document Merging

- **Keyword Counting Setup:** Split text, count keywords, then aggregate
- **Document Merging Setup:** Combine NDAs into one with max info & min redundancy



Latency vs Volume: A New Metric

- **Volume:** Number of thoughts that contribute to a final thought
- **Latency:** Number of steps (hops) to reach final answer

Scheme	Latency	Volume
CoT	N	N
CoT-SC	N/k	N/k
ToT	log□N	O(log□N)
GoT	log□N	**N**

GoT achieves **low latency and high volume**

Summary

Feature / Capability	Chain-of-Thought (CoT)	Tree-of-Thought (ToT)	Graph-of-Thought (GoT)
Structure Type	Linear (chain)	Tree (branching paths)	Arbitrary graph (general DAG)
Exploration	Single path	Multiple paths	Arbitrary path combinations
Backtracking	❌	✅ (via DFS)	✅
Aggregation of Thoughts	❌	❌	✅
Thought Refinement / Looping	❌	Partial (manual)	✅ (loop edges)
Emergent Reasoning	Yes (large models only)	Stronger via structure	Most expressive + modular
Implementation Complexity	Low	Moderate	High
Ideal For	Simple logic, math	Exploratory, multi-step	Complex, multi-modal workflows

As tasks become more elaborate, moving from chains to trees to graphs offers increasing **flexibility, modularity, and alignment with how humans reason.**

From Explicit Steps to Latent Spaces: The Evolution of Reasoning in Language Models

Lu Li
April 17

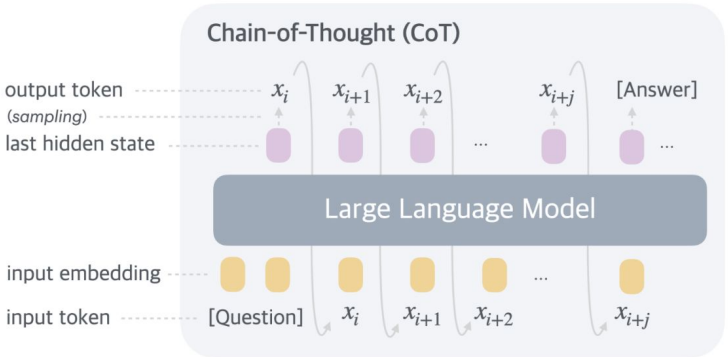
Implicit CoT with stepwise internalization

From Explicit CoT to Implicit CoT: Learning to Internalize CoT Step by Step

Yuntian Deng[1,2] **Yejin Choi**[1,3] **Stuart Shieber**[4]
[1]Allen Institute for Artificial Intelligence [2]University of Waterloo
[3]University of Washington [4]Harvard University
{yuntiand, yejinc}@allenai.org, shieber@seas.harvard.edu

Motivation

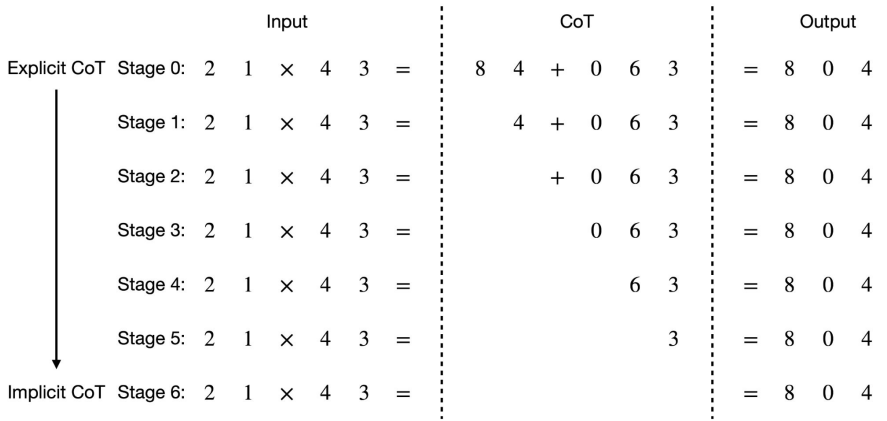
- Explicit CoT improves accuracy but increases inference latency
  - e.g., 9×9 multiplication requires 246 CoT tokens.
- Limitations of Existing Work:
  - Explicit CoT: Slow due to long reasoning chains.
  - No CoT: Fails on complex tasks (e.g., GPT-4o scores 0% on 9×9 multiplication).
- Goal: Achieve implicit CoT—retain CoT's accuracy without generating intermediate steps.



Method: Stepwise Internalization

- **Idea:** Gradually remove CoT tokens during training to internalize reasoning.
- **Training Process:**
  - **Start**: Train a model using Explicit CoT (predicts all intermediate steps z1, …., zm, and the final answer y
  - **Gradually Remove**: Finetune the model in stages. In each stage, remove one (or more) CoT token(s) from the *beginning of the sequence* the model needs to predict.
    - Stage 0: predict z1, z2, …, zm, y
    - Stage 1: predict z2, …, zm, y
    - …
    - Final stage: predict only y (all zi removed)
- Finetune until all CoT steps are internalized.

$$12 \times 4 + 12 \times 30 = \underbrace{48}_{\text{reversed: } 84} + \underbrace{360}_{\text{reversed: } 063} .$$



Method

- **Original loss function:**

$$\min_{\theta} -\log P_\theta(y, z_{1:m} \mid x),$$

- **At each step t of the training process, remove (up to) s(t) tokens from the intermediate steps z. Update loss function:**

$$\min_{\theta} -\log P_\theta(y, z_{1+\min(s(t),m):m} \mid x).$$

- **Linear schedule for removing tokens:** $s(t) = \left\lfloor \Delta \frac{t}{T} \right\rfloor$

  where T is the total number of steps per epoch, and Δ is a hyperparameter controlling how many CoT tokens are removed per epoch.

Method - Ensuring Stability

- **Challenge:** instability in the training process due to changes in the loss function over time
- **Potential reasons:**
  - A sudden change in the loss function, caused by the removal of one more CoT token, results in abrupt changes in the second-order gradients
    - **Solution: Optimizer Reset.** When removing the next CoT token(s) from the target, reset the optimizer's internal state (e.g., momentum in AdamW). Prevents issues from old gradient estimates.
  - Model lacks adaptability to removal of more tokens
    - **Solution: Removal Smoothing.**
    - Introduce a small probability of removing more tokens than scheduled for the current stage.
    - Uses a hyperparameter λ to control the probability distribution (e.g., λ=4 means 98% chance of removing the scheduled number, 2% chance of removing more).
    - Helps the model "prepare" for the next stage, smoothing the transition.

$$s(t)^* = s(t) + o, \qquad P(o) \propto \exp(-\lambda o).$$

Experiment setup

- **Tasks:**
  - **Multi-digit Multiplication** (4×4 to 9×9 digits).
  - **GSM8K** (grade-school math problems).
- **Models:**
  - Small-scale: GPT-2 Small (117M).
  - Large-scale: Mistral 7B.
- **Baselines:**
  - No CoT, Explicit CoT, ICotKD (Deng et al., 2023).

Results – Multi-digit Multiplication

- Multi-digit multiplication.
- 9×9 Multiplication:
  - 99% accuracy (vs. 0% for No CoT).
  - **Speed**: Matches No CoT (1.0× speed), 11× faster than Explicit CoT.
- Comparison to Larger Models:
  - **Outperforms MathGLM-2B** (90% on 5×5 vs. ICotSI's 95%).

Model	4×4		5×5		7×7		9×9	
	Acc	Speed	Acc	Speed	Acc	Speed	Acc	Speed
GPT-2 Small (117M)								
Explicit CoT	1.00	0.17	1.00	0.14	1.00	0.12	1.00	0.09
No CoT	0.29	1.00	0.01	1.00	0.00	1.00	0.00	1.00
ICoT-KD	0.97	0.67	0.10	0.71	-	-	-	-
ICoT-SI	1.00	1.02	0.95	1.00	0.95	1.00	0.99	1.00
MathGLM-100M								
No CoT	0.80	1.00	0.56	1.00	-	-	-	-
MathGLM-500M								
No CoT	0.90	1.00	0.60	1.00	-	-	-	-
MathGLM-2B								
No CoT	0.95	1.00	0.90	1.00	-	-	-	-
GPT-3.5†								
Explicit CoT	0.43	0.10	0.05	0.07	0.00	0.15	0.00	0.11
No CoT	0.02	1.00	0.00	1.00	0.00	1.00	0.00	1.00
GPT-4†								
Explicit CoT	0.77	0.14	0.44	0.14	0.03	0.09	0.00	0.07
No CoT	0.04	1.00	0.00	1.00	0.00	1.00	0.00	1.00

Results - Grade School Math (GSM8K)

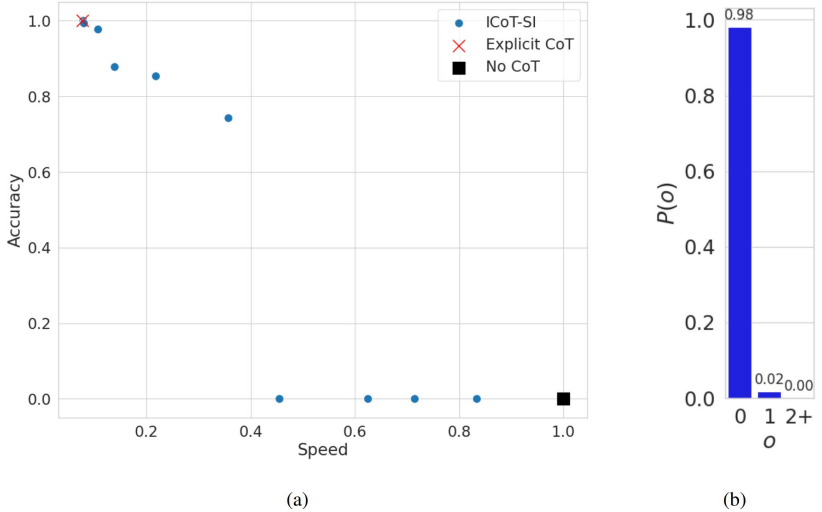
- Effective beyond arithmetic.
- Accuracy vs. Speed Trade-off:
  - Explicit CoT is still more accurate (e.g., Mistral-7B gets 68%).
  - But ICoT-SI provides a strong accuracy level at much higher speeds (No CoT speed).

Table 3: Accuracy of various approaches on GSM8K. †: 5-shot prompted instead of finetuned.

Model	GPT-2 Small	GPT-2 Medium	Phi-3 3.8B	Mistral 7B	GPT-3.5†	GPT-4†
Explicit CoT	0.41	0.44	0.74	0.68	0.62	0.91
No CoT	0.13	0.17	0.28	0.38	0.03	0.44
ICoT-KD	0.20	0.22	-	-	-	-
ICoT-SI	0.30	0.35	0.31	0.51	-	-

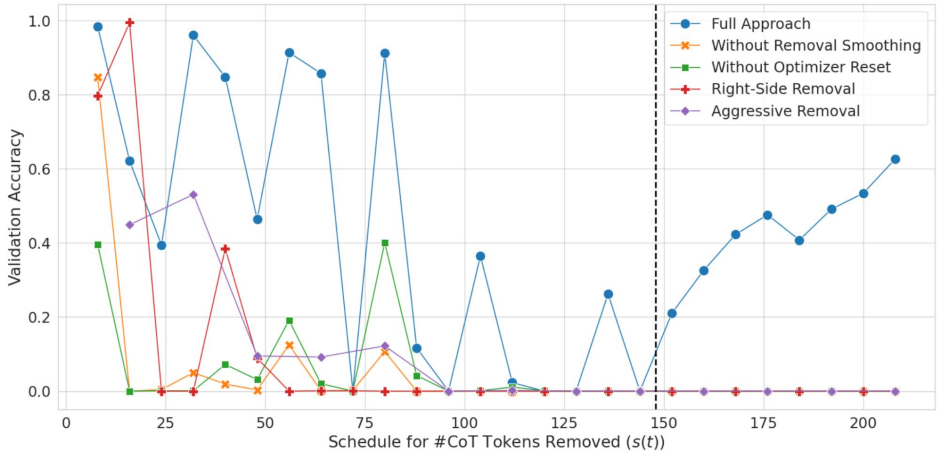
Accuracy - speed trade-off

- Partial Internalization: ICoT-SI allows stopping the process before all CoT tokens are removed.
- Benefit: Creates a trade-off curve. You can choose a point that balances desired speed and acceptable accuracy.
- Example (11x11 Multiplication): Even if full internalization isn't perfect, internalizing some steps can yield >70% accuracy at 4x the speed of full Explicit CoT.



Ablation Studies

- Experiments confirm the necessity of the proposed stabilization methods.
  - Without Removal Smoothing: Training becomes unstable and fails to recover accuracy.
  - Without Optimizer Reset: Training becomes unstable and fails.
- Other Findings:
  - Removal Side: Removing tokens from the beginning (left) is crucial. Removing from the end performs poorly.
- Removal Rate (Δ): Removing too many tokens per epoch ("Aggressive Removal") leads to instability and failure. Needs carefu



Discussion

Limitations

- Training Cost: Finetuning repeatedly can be expensive, especially for tasks with very long CoT chains.
- Training Instability: Requires careful tuning of the removal rate (Δ); can fail if too aggressive.
- Interpretability: Like No CoT, the final model lacks explicit reasoning steps (though hidden states could potentially be probed).
- Accuracy Gap: Explicit CoT still generally achieves the highest accuracy, although ICoT closes the gap significantly compared to No CoT while matching its speed.

Strength

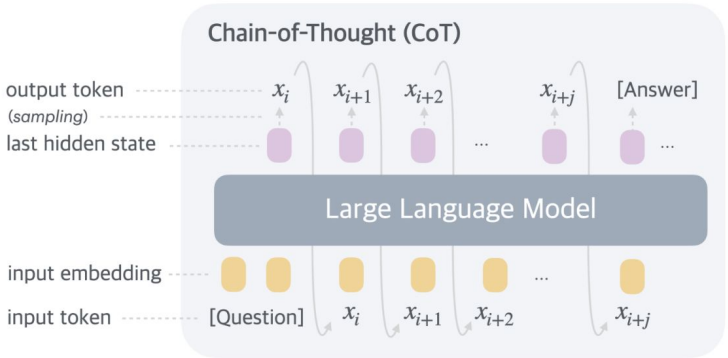
- Achieves strong results on multiplication and GSM8K, significantly outperforming No CoT, at No CoT inference speed.

Training LLMs to reason in a continuous latent space

- the potential of **LLM reasoning in an unrestricted latent space** instead of using natural language
- Coconut (Chain of Continuous Thought)

Motivation

- Can we free the reasoning from being within the language space
- Instead of mapping between hidden states and language tokens using the language model head and embedding layer, Coconut directly feeds **the last hidden state (a continuous thought)** as the input embedding for the next token



Method

$$H_t = \text{Transformer}(E_t)$$

$$\mathcal{M}(x_{t+1} \mid x_{\leq t}) = \text{softmax}(Wh_t)$$

- The standard LLM can be described using
- In language mode, the model operates as a standard language model, autoregressively generating the next token.
- In latent mode, it directly utilizes the last hidden state as the next input embedding.
- This last hidden state represents the current reasoning state, termed as a “continuous thought”.

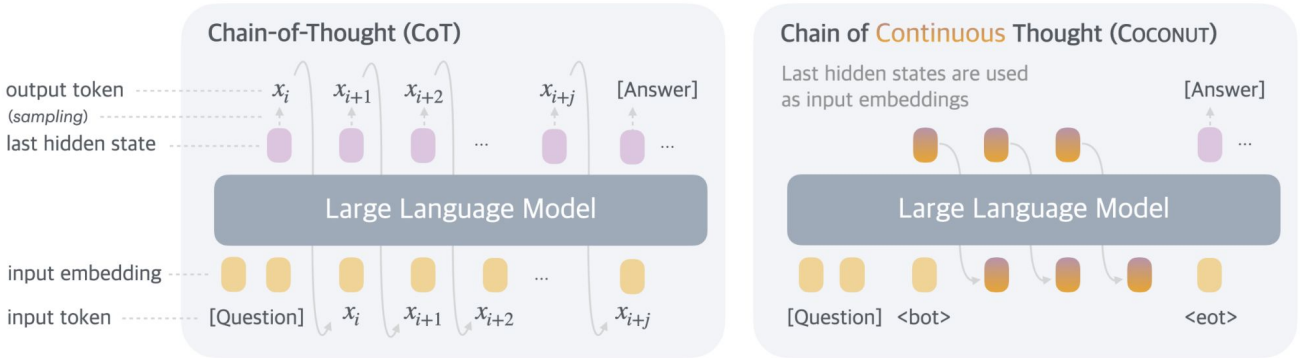


Figure 1 A comparison of Chain of Continuous Thought (Coconut) with Chain-of-Thought (CoT). In CoT, the model generates the reasoning process as a word token sequence (e.g., $[x_i, x_{i+1}, ..., x_{i+j}]$ in the figure). Coconut regards the last hidden state as a representation of the reasoning state (termed “continuous thought”), and directly uses it as the next input embedding. This allows the LLM to reason in an unrestricted latent space instead of a language space.

Method

- During latent mode (i < t < j), use the last hidden state from the previous token to replace the input embedding $E_t = [e(x_1), e(x_2), ..., e(x_i), h_i, h_{i+1}, ..., h_{t-1}]$
- After the latent mode finishes, the input reverts to using the token embedding, i.e.,

$$E_t = [e(x_1), e(x_2), ..., e(x_i), h_i, h_{i+1}, ..., h_{j-1}, e(x_j), ..., e(x_t)]$$

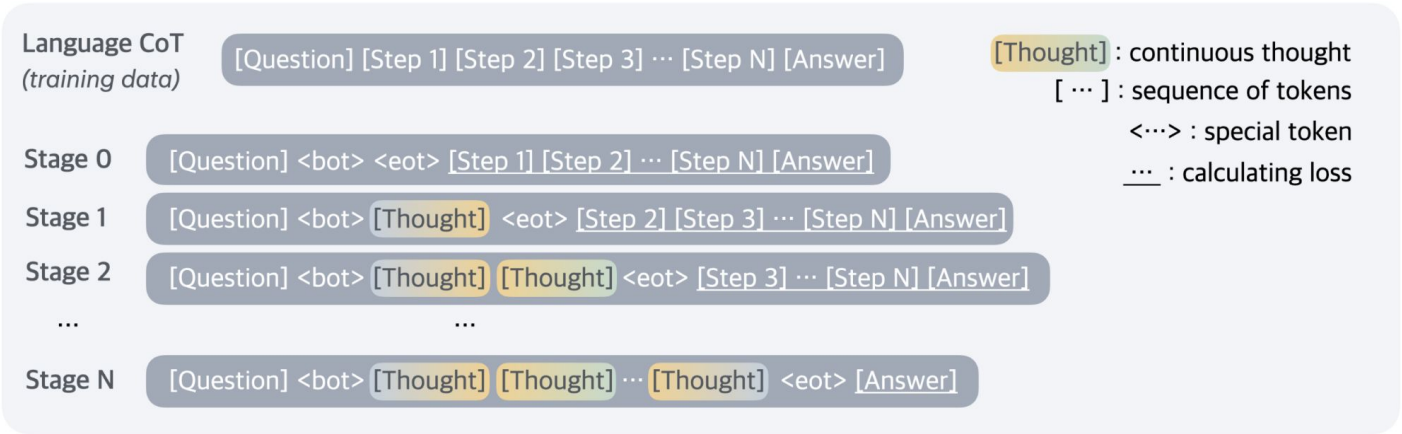


Figure 2 Training procedure of Chain of Continuous Thought (COCONUT). Given training data with language reasoning steps, at each training stage we integrate c additional continuous thoughts ($c = 1$ in this example), and remove one language reasoning step. The cross-entropy loss is then used on the remaining tokens after continuous thoughts.

Training Procedure

- In stage 0, the model is trained on regular CoT instances.
- At the k-th stage, the first k reasoning steps in the CoT are replaced with k × c continuous thoughts, where c is a hyperparameter controlling the number of latent thoughts replacing a single language reasoning step
- This process is **fully differentiable** and allows for back prop

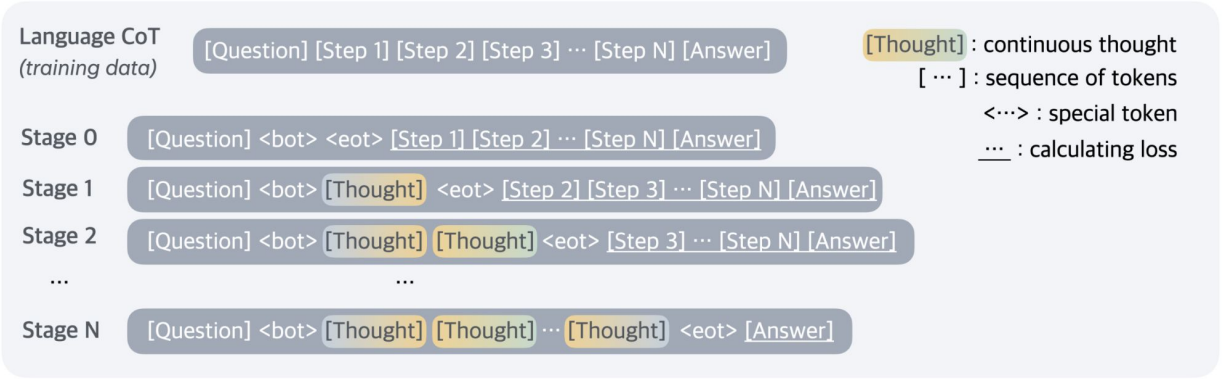


Figure 2 Training procedure of Chain of Continuous Thought (COCONUT). Given training data with language reasoning steps, at each training stage we integrate c additional continuous thoughts ($c = 1$ in this example), and remove one language reasoning step. The cross-entropy loss is then used on the remaining tokens after continuous thoughts.

Inference process

- Same as standard language decoding, except in the latent mode, feed the last hidden state as the next input embedding
- How to switch between language mode and latent mode?
  - For <bot>, directly after question tokens
  - For <eot>,
    - a) train a binary classifier on latent thoughts to enable the model to autonomously decide when to terminate the latent reasoning, or
    - b) always pad the latent thoughts to a constant length.
  - Both approaches work comparably well. Therefore, used the second option in experiment for simplicity, unless specified otherwise.

Experiment setup

- Math: GSM8k
- Logic: 5-hop ProntoQA + ProsQA
- **Baselines:**
  - **CoT:** Standard supervised finetuning on full language reasoning chains.
  - **No-CoT:** Train to predict answer directly from question.
  - **iCoT:** Internalized CoT via gradual removal of language steps (related training strategy).
  - **Pause Token:** Insert special <pause> tokens instead of language steps (provides compute budget but no feedback loop like CoCoT).
- **CoCoT Ablations:**
  - **w/o curriculum:** Train CoCoT directly on final stage data (no gradual steps).
  - **w/o thought:** Use the multi-stage curriculum but *without* inserting continuous thoughts (similar to iCoT but with CoCoT's schedule).
  - **pause as thought:** Use the multi-stage curriculum but insert <pause> tokens instead of continuous thoughts.

Results

- Saves the number of tokens needed and thus saves inference time
- CoCoT Effectiveness: Consistently improves accuracy over No-CoT.
- Planning Tasks (ProsQA): Latent reasoning methods show a clear advantage over CoT
- Importance of Curriculum: Training CoCoT without the multi-stage curriculum performs poorly (similar to No-CoT). Guidance is needed.

Method	GSM8k		ProntoQA		ProsQA	
	Acc. (%)	# Tokens	Acc. (%)	# Tokens	Acc. (%)	# Tokens
CoT	42.9 ±0.2	25.0	98.8 ±0.8	92.5	77.5 ±1.9	49.4
No-CoT	16.5 ±0.5	2.2	93.8 ±0.7	3.0	76.7 ±1.0	8.2
iCoT	30.0*	2.2	99.8 ±0.3	3.0	98.2 ±0.3	8.2
Pause Token	16.4 ±1.8	2.2	77.7 ±21.0	3.0	75.9 ±0.7	8.2
COCONUT (Ours)	34.1 ±1.5	8.2	99.8 ±0.2	9.0	97.0 ±0.3	14.2
- *w/o curriculum*	14.4 ±0.8	8.2	52.4 ±0.4	9.0	76.1 ±0.2	14.2
- *w/o thought*	21.6 ±0.5	2.3	99.9 ±0.1	3.0	95.5 ±1.1	8.2
- *pause as thought*	24.1 ±0.7	2.2	100.0 ±0.1	3.0	96.6 ±0.8	8.2

Results

- As we increased c from 0 to 1 to 2, the model's performance steadily improved
- Suggesting a "chaining" benefit similar to CoT, but in latent space

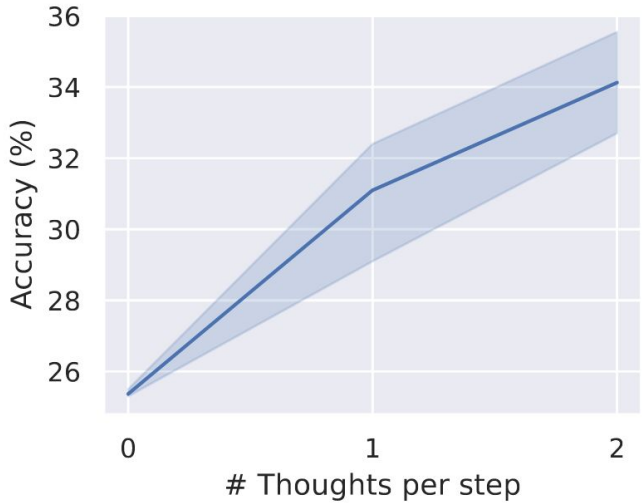
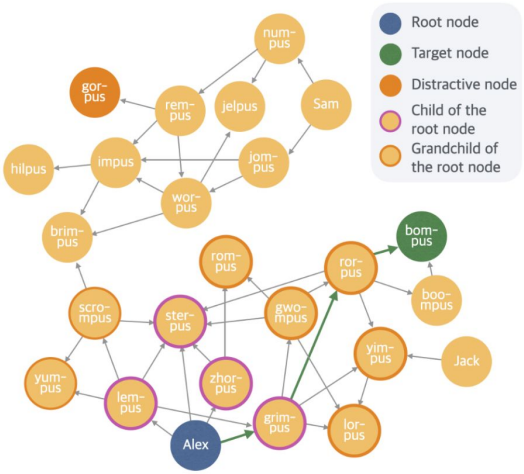


Figure 3 Accuracy on GSM8k with different number of continuous thoughts.

Experiments - Latent Reasoning as Search

- **Question**: Why does CoCoT excel, especially in planning tasks like ProsQA?
- **Hypothesis**: Continuous thoughts aren't just single steps; they can encode multiple possibilities simultaneously.
- **Interpretation**: Latent reasoning in CoCoT behaves like a form of implicit BFS or tree search.
  - The model doesn't commit to one path prematurely.
  - It explores multiple options within the continuous thoughts.
  - It uses implicit value functions (observed via token probabilities when forced back to language) to prune bad paths over subsequent thought steps.



Question:

Every grimpus is a yimpus. Every worpus is a jelpus. Every zhorpus is a sterpus. Alex is a grimpus ⋯ Every lumps is a yumpus.
Question: **Is Alex a gorpus or bompus?**

Ground Truth Solution

Alex is a grimpus.
Every grimpus is a rorpus.
Every rorpus is a bompus.
Alex is a bompus

CoT

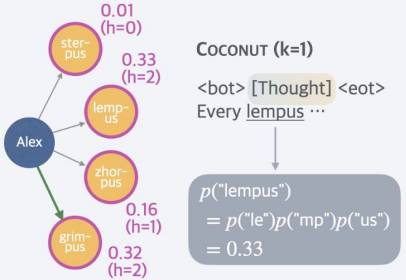
Alex is a lempus.
Every lempus is a scrompus.
Every scrompus is a yumpus.
Every yumpus is a rempus.
Every rempus is a gorpus.
Alex is a gorpus ✗
(Hallucination)

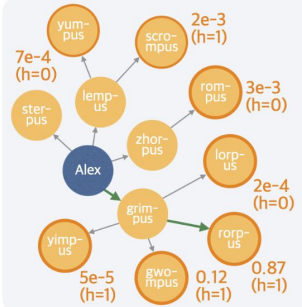
Coconut (k=1)

<bot> [Thought] <eot>
Every lempus is a scrompus.
Every scrompus is a brimpus.
Alex is a brimpus ✗
(Wrong Target)

Coconut (k=2)

<bot> [Thought] [Thought] <eot>
Every rorpus is a bompus.
Alex is a bompus ✓
(Correct Path)





Why Latent Space Helps Planning

- **Analysis Method**: Interpolate between latent and language reasoning (vary k initial latent steps) on ProsQA.
- Accuracy improves, and errors (Hallucination, Wrong Target) decrease as more reasoning happens in latent space (higher k).
- Better planning ability when more reasoning happens in the latent space.

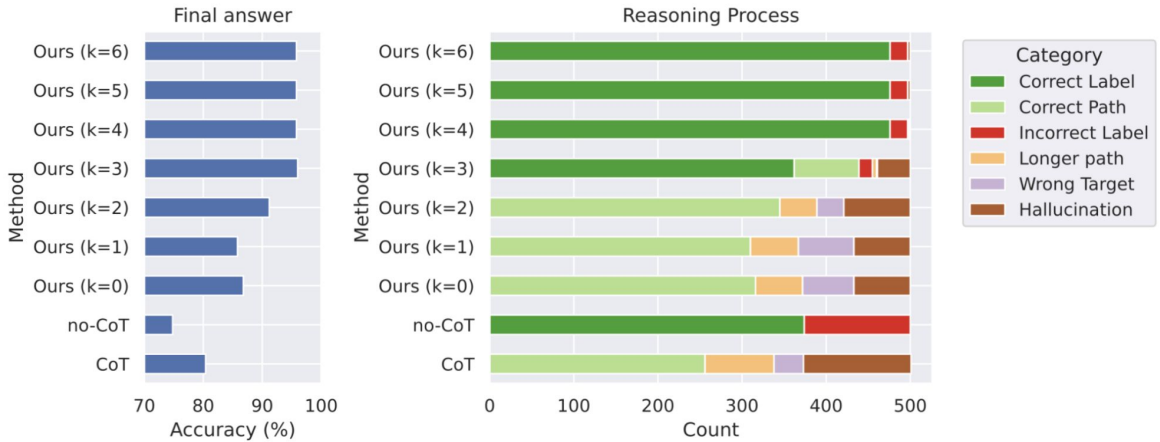
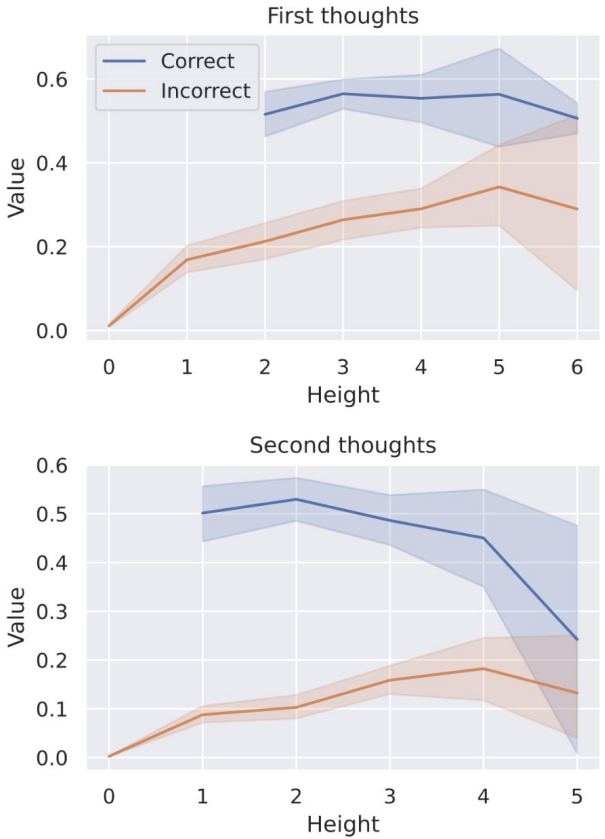


Figure 5 The accuracy of final answer (left) and reasoning process (right) of multiple variants of COCONUT and baselines on ProsQA.

Why Latent Space Helps Planning

- It's easier for the model to evaluate the "value" of nodes closer to the end of a potential reasoning path (lower "height" in the search tree).
- **Takeaway**:
  - Language CoT forces early, hard decisions when node values are uncertain.
  - CoCoT's latent reasoning delays these hard decisions.
  - By exploring in latent space, the model effectively pushes evaluation closer to the "leaves" of the search tree, where distinguishing good/bad paths is easier.



Case Study Snippet

- Continuous thoughts are efficient representations of reasoning.
- CoT Trace of typical language steps
  - 3 × 3 × 60 = 9 × 60 = 540, or 3 × 3 × 60 = 3 × 180 = 540
- Continuous thought:
  - Decode the first continuous thought
  - Highlight: The single continuous thought captures multiple potential first steps or key variables needed for different valid reasoning traces.

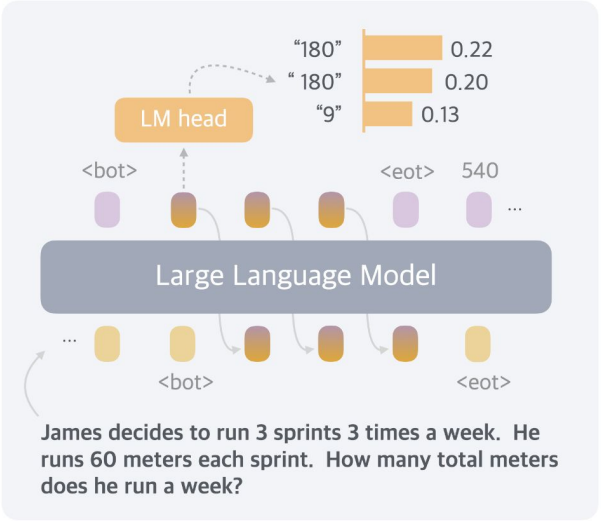


Figure 4 A case study where we decode the continuous thought into language tokens.

Discussion

Strengths:

- Introduced CoCoT, a paradigm for LLM reasoning in continuous latent space.
- Demonstrated effectiveness, especially on planning-intensive tasks, often with higher efficiency (fewer tokens) than CoT.
- Latent reasoning is like BFS, can improve inference speed and improve performance on tasks that require planning

Limitations:

- Training efficiency (multiple forward passes).
- Interpretability is sacrificed
- Not performing well on GSM8k

Thank you!