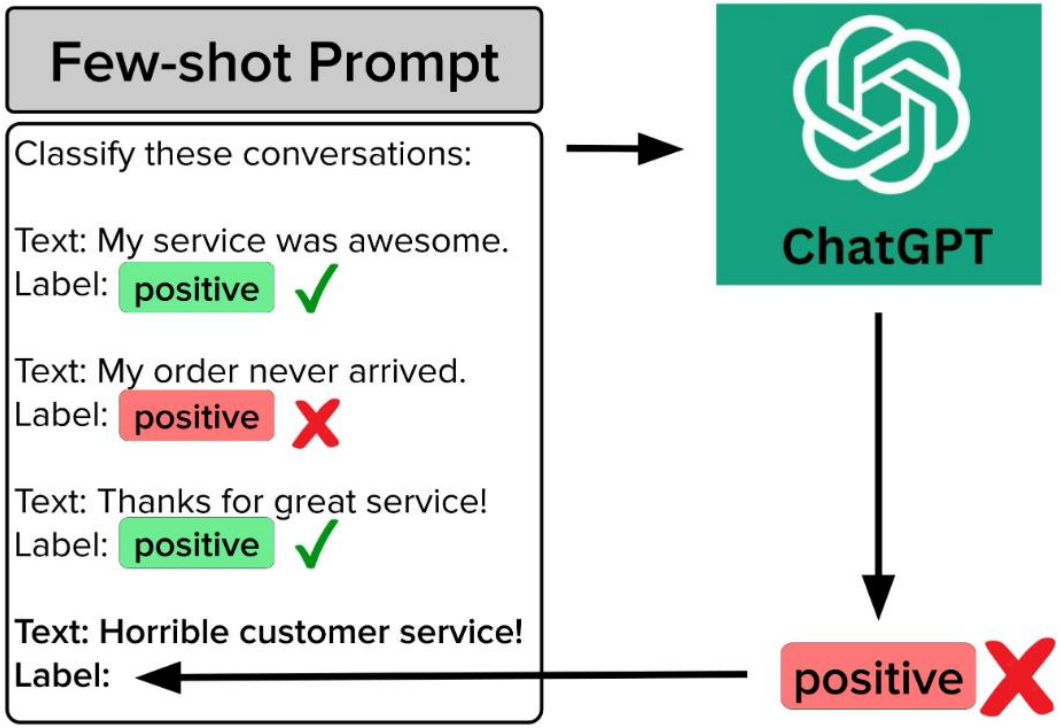
Why Prompt Design Matters and Works: A Complexity Analysis of Prompt Search Space in LLMs

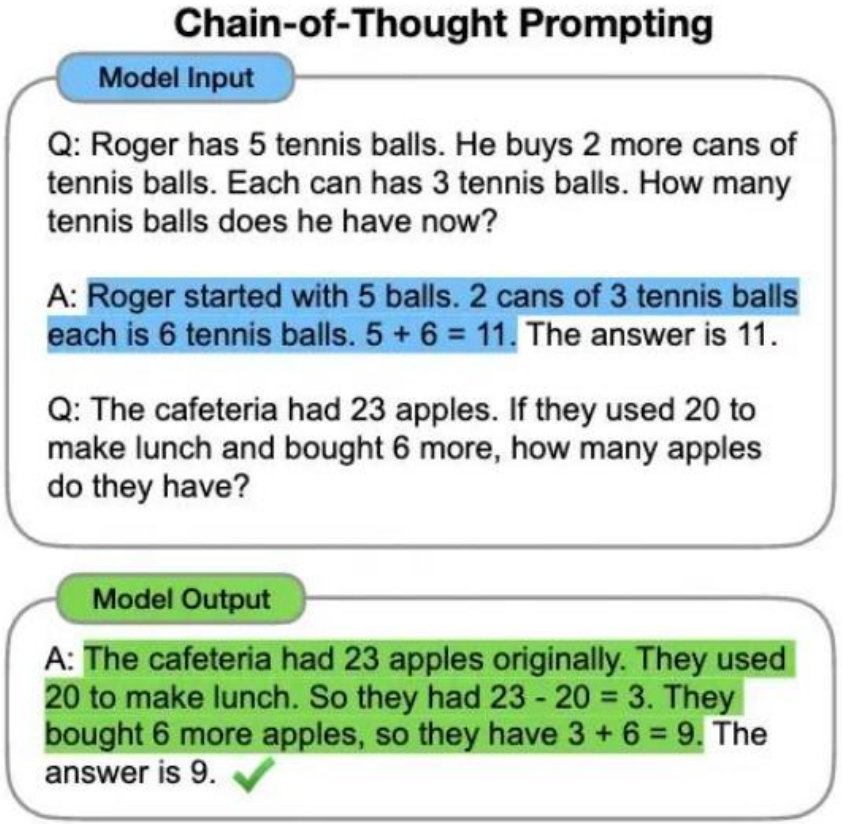
Xiang Zhang[1*] **Juntai Cao**[1*] **Jiaqi Wei**[3] **Chenyu You**[2†] **Dujian Ding**[1†]
[1]University of British Columbia, [2]Stony Brook University, [3]Zhejiang University
xzhang23@ualberta.ca, {jtcao7, dujian}@cs.ubc.ca,
jiaqi.wei@zju.edu.cn, cyou@cs.stonybrook.edu

ACL 2025

Prompt

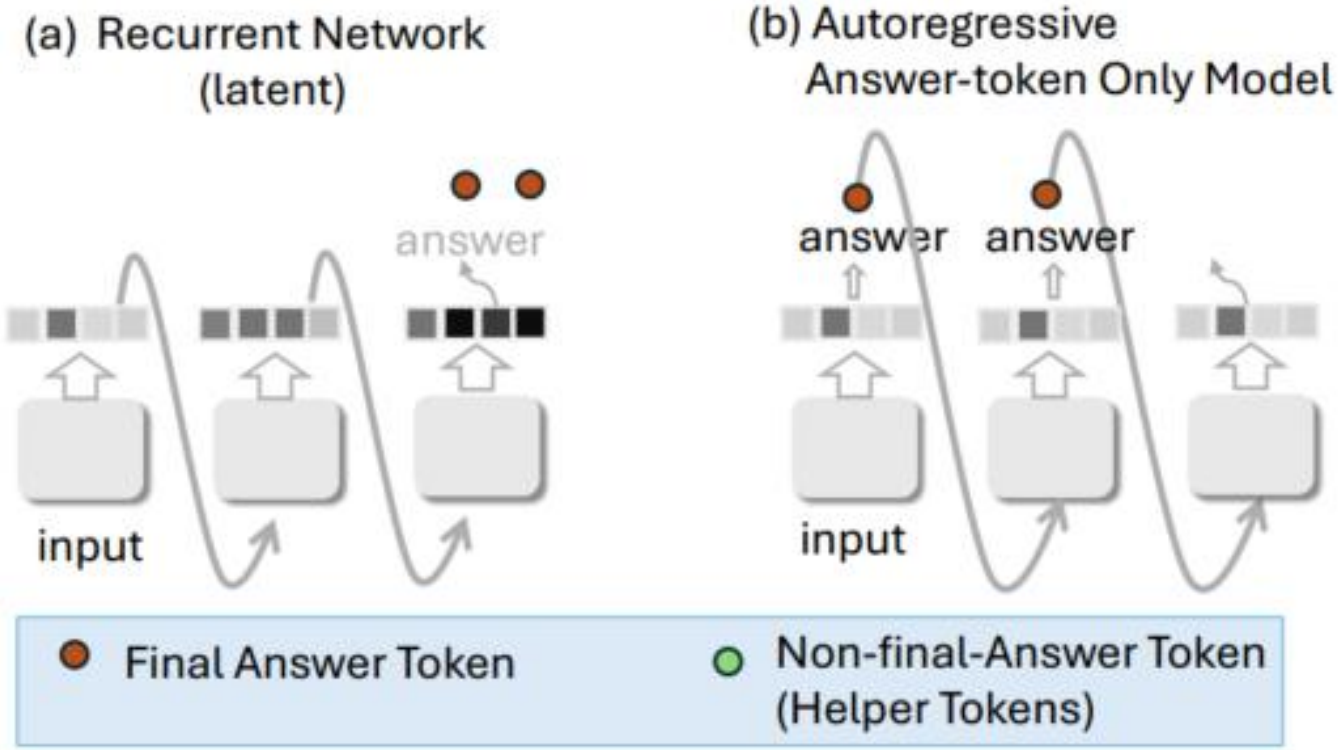
Prompt: An instruction, question, or input text that a user gives to an AI model to clearly tell the model what task needs to be completed and how to output the results.





- Why does a particular prompt design work?
- How to find the optimal prompt desigh?

Limitations of Transformer Architecture



h stores reasoning memory and intermediate reasoning results
the ability to sequentially compute and update ***h*** over time
allows a model to build reasoning depth

Answer ***y*** is a discrete value extracted from ***h*** and only contains partial information
y exists outside the latent space where ***h*** operates, meaning it cannot be used for computation

RNN: Sequential token processing

$$\boldsymbol{h_t = g_\theta(h_{t-1}, x_t)}$$

This kind of 'recursive transmission' can naturally accumulate temporal information

Transformer: Self-Attention

$\boldsymbol{h_t}$ =Attention-weighted sum of all tokens

For RNN: The longer the sequence, the more times the hidden state is recursively updated, and the inference depth naturally increases.

For Transformer: The computation of its hidden state depends only on the 'number of layers' of the model and is independent of the length of the input sequence.

Nature of Inductive Reasoning

$$x_n = (x_1, x_2, \ldots, x_n)$$

To validate the n-th move, the n-th board state $\boldsymbol{h_n}$ must be calculated.

This requires n sequential computations, as the n-th board state depends not only on the sequence of moves $\boldsymbol{x}$ but also on the previous board state $\boldsymbol{h_{n-1}}$.

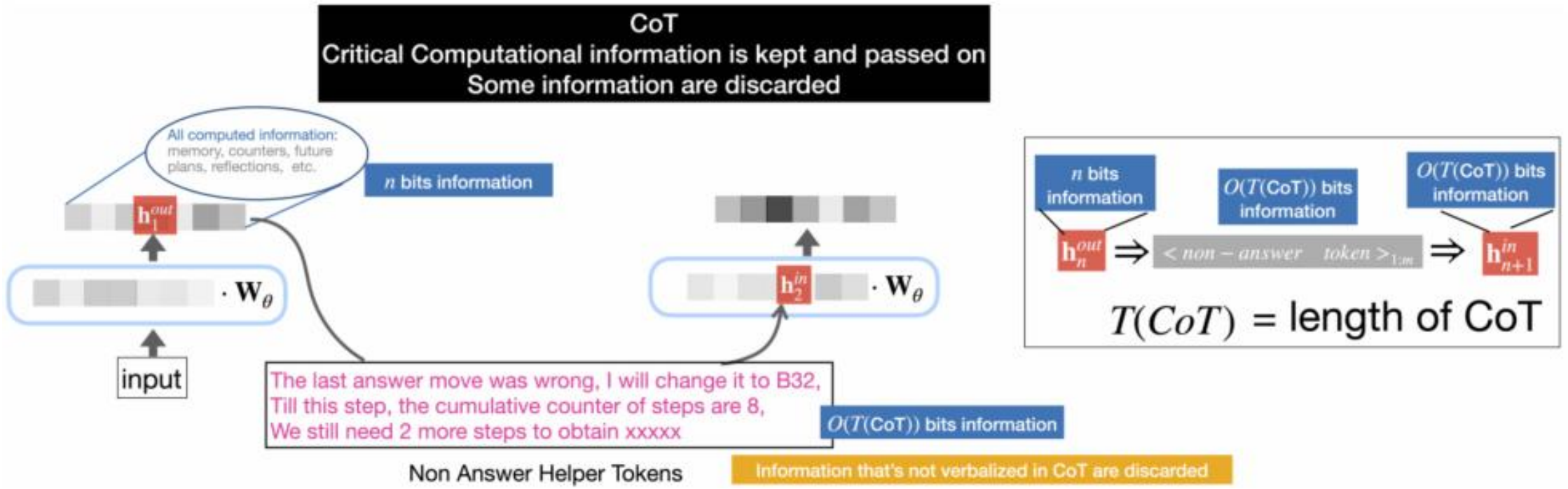
While a neural network could memorize the mapping from $\boldsymbol{x_{1:n}}$ to the correct $\boldsymbol{h}$, the memorization will be an exponentially growing challenge

Reasoning inherently requires sequential depth

To solve a given task, there is a theoretical lower bound on the required depth of computation

Transformer's fixed sequential reasoning depth over the hidden state $\boldsymbol{h}$ prevents them from solving tasks that require deeper reasoning as input length grows

CoT+Autoregressive=Recurrent

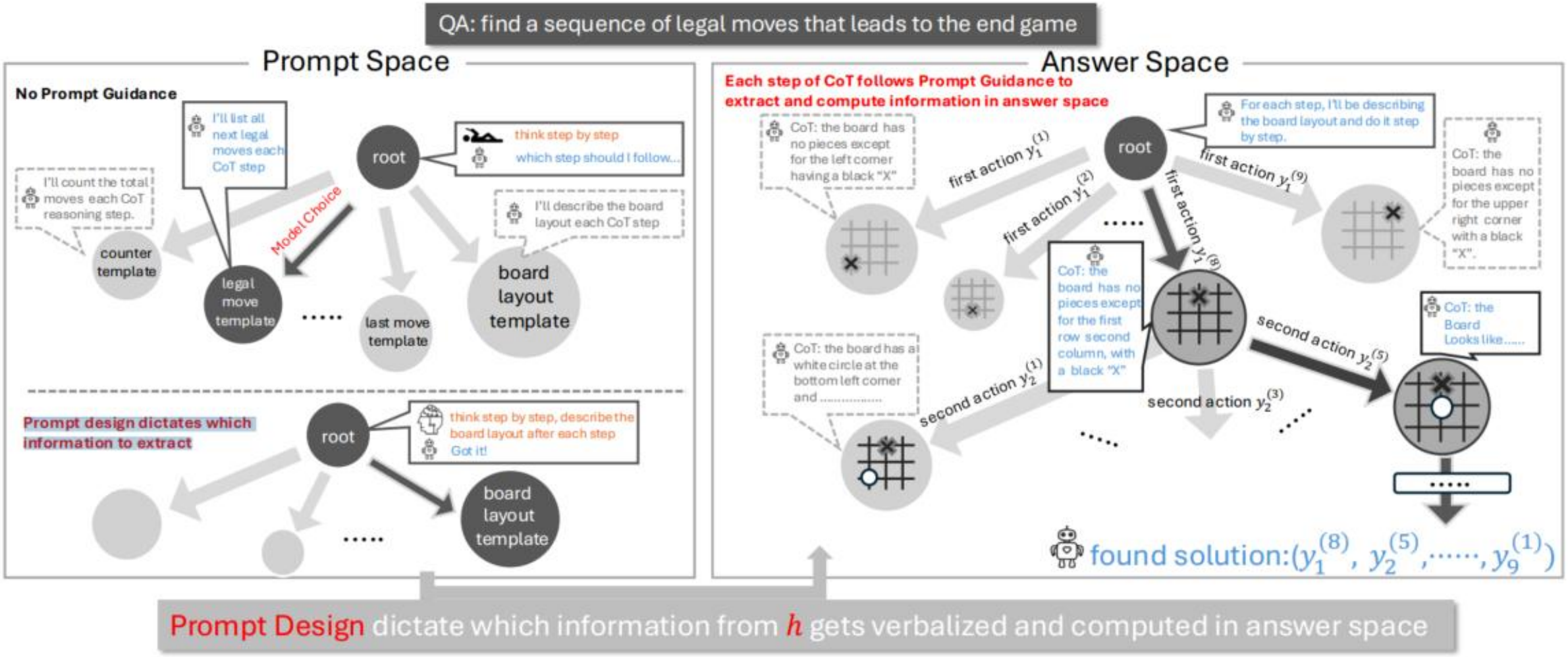


CoT extends beyond simple answer token generation by producing intermediate steps as non-answer natural language tokens $(\boldsymbol{o_1}, \boldsymbol{o_2}, \ldots, \boldsymbol{o_k})$ that act as a discretizations of latent information $\boldsymbol{h_n}$

$$\mathbf{h}_t \xrightarrow{\text{discritization}} (\mathrm{o}_1, \mathrm{o}_2, \cdots, \mathrm{o}_\mathrm{k}) \xrightarrow{\text{vectorization}} \mathbf{h}_{t+1}$$

LLMs with CoT extend reasoning from latent space $\boldsymbol{H}$ to natural language token space $\boldsymbol{O}$.

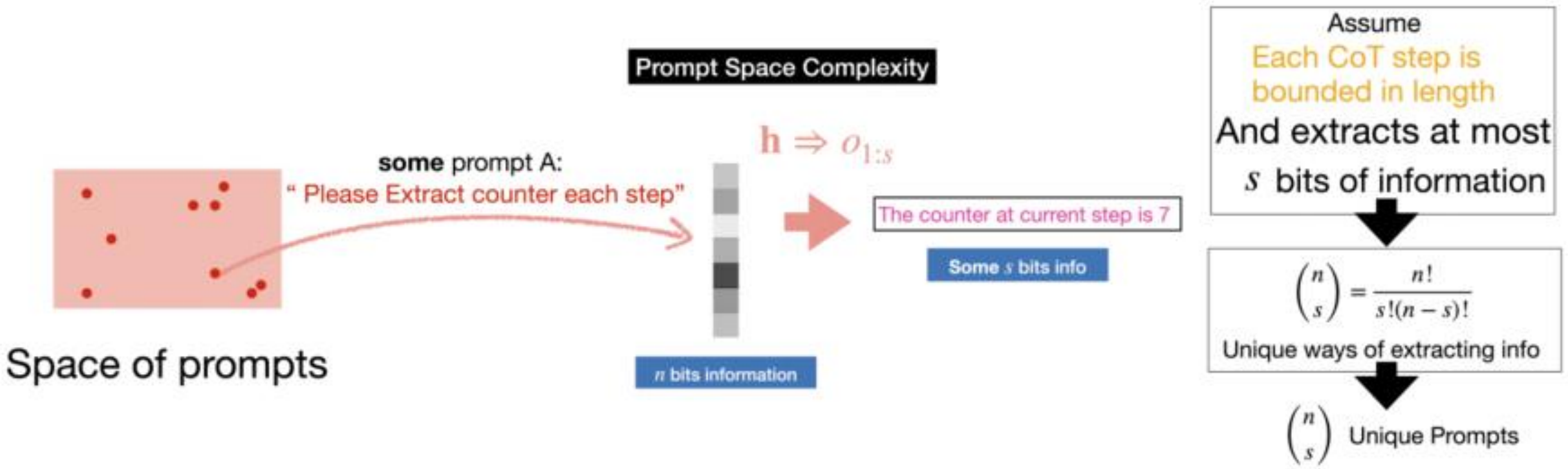
CoT Search Space = Prompt Space + Answer Space



Decompose the CoT reasoning into two components:

1. template search within the prompt space
2. answer search within the answer space

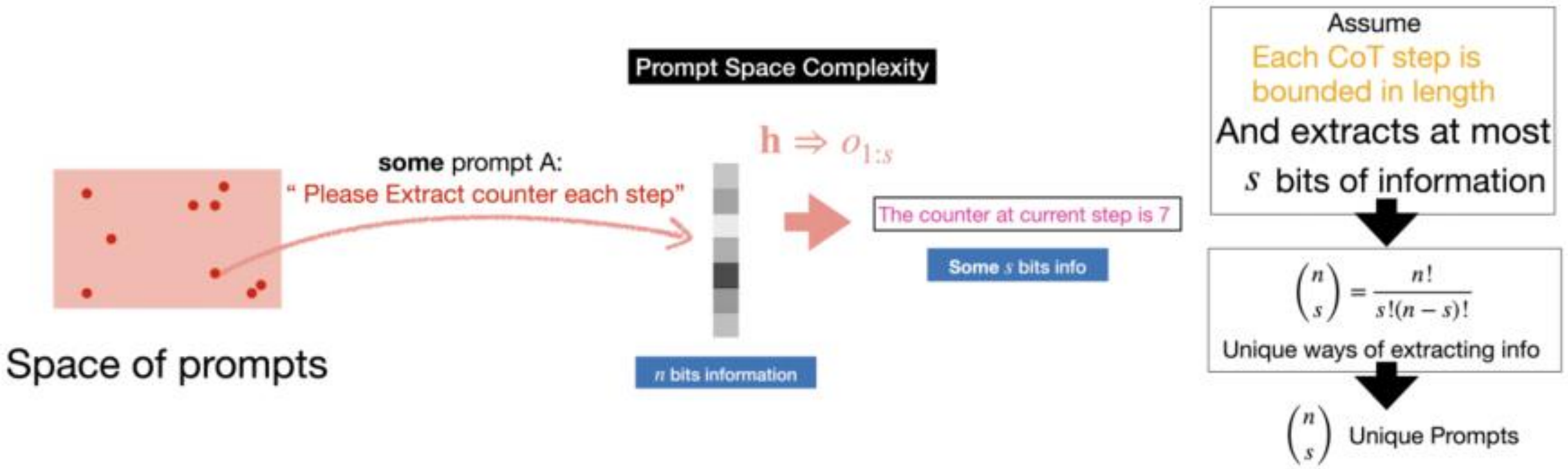
Prompt Space Complexity



When LLMs are prompted to perform tasks, they follow a step template, specifying which information from ***h*** to extract and discretize into non-answer helper tokens $(o_1, o_2, \ldots, o_{T(CoT)})$ in CoT.

Ideally, as $T_{(CoT)} -> \infty$——meaning the length of the each CoT step is arbitrarily long——all vectorized information in ***h*** can be fully textualized

Prompt Space Complexity



Define the amount of information stored in ***h*** as ***n*** bits, and each CoT step extracts up to ***s*** bits of information into ***o***, each unique step template specifies a way to extract ***s*** bits from the full n-bit space

$$C(n,s) = \frac{n!}{s!\,(n!-s!\,)}$$

The prompt template defines how information is extracted and used recurrently in the CoT process.
Finding the correct template is equivalent to discovering the algorithm for solving a given task

Answer Space Complexity

With a specific step (prompt) template p_i chosen from the prompt space ***P***, CoT iteratively executes $h_t \overset{p_i}{\Rightarrow} (o_1^{(i)}, o_2^{(i)}, \ldots, o_k^{(i)}) \Rightarrow h_{t+1}$ to update ***h*** and calculate the next state, continuing this process until reaching the final state (solution).

In the chess simulation task of <*finding a set of actions leading to game end*>:

the answer space $S = (s_1, s_2, \ldots, s_\infty)$ contains all possible combinations of action sequences *s*

The solution set ***CR*** $\subset$ ***S*** includes all valid action sequences that lead to the end of the game, being a subset of the entire answer space ***S***

Solving the problem requires identifying one single correct action sequence $s_{correct} = (y_1, y_2, \ldots, y_T) \in CR$

The complexity of navigating the answer space can be roughly measured by

$$\frac{len(CR)}{len(S)} \mid p$$

The size of answer space *S*: the more open the task, the more possible outcomes, and the larger *S* is.

The size of correct solution set *CR*: he stricter the task, the fewer correct solutions there are, and the smaller the *CR*.

The prompt template (*p*) directly determines which hidden state information the model 'extracts,' thereby reshaping the complexity of the answer space *S*.

Experiment

Level	Task	RNN	Tape RNN	Transformer	LLM w/o CoT	CoT Unsupervised	CoT Supervised	CoT Supervised-SUB
R	Modular Arithmetic	**1.00**	**1.00**	**0.96**	0.22	**0.96**	**1.00**	0.44
	Parity Check	**1.00**	**1.00**	0.52	0.58	**0.94**	**1.00**	0.42
	Cycle Navigation	**1.00**	**1.00**	0.62	0.50	0.78	**1.00**	0.26
CF	Stack Manipulation	0.56	1.00	0.58	0.00	**0.92**	**0.96**	0.00
	Reverse List	0.62	**1.00**	0.62	0.48	0.80	**0.96**	0.38
	Modular Arithmetic	0.41	0.95	0.32	0.00	0.82	**0.94**	0.50

Solving these tasks demands a minimum computational depth that scales linearly with input length, surpassing the constant depth inherent to Transformer models.
R represents simple recursive tasks, **CF** represents complex recursive tasks
RNN/Tape-RNN represents models with inherent recursive capabilities
Transformer represents a pure Transformer model without CoT assistance
CoT (Unsupervised) simply means 'Let's think step by step'
CoT (Supervised) refers to the optimal prompt template, directly defined the core algorithm of the task, allowing the model to focus directly on effective reasoning paths without independent exploration.
CoT (Supervised-SUB) represents a suboptimal prompt template, extracting irrelevant information or designing redundant steps, causing the model to be unable to effectively propagate intermediate states, and even being misled into incorrect reasoning paths

Experiment

Model	R			CF		
	MA	PC	CN	SM	RL	MA
Unsupervised **CoT**	**0.96**	**0.94**	0.78	0.92	0.80	0.82
Unsupervised **ToT**	**0.92**	**0.90**	**0.92**	0.36	0.88	0.78
Unsupervised **GoT**	**1.00**	**0.98**	**0.90**	0.72	**0.92**	0.88
Correctly supervised **CoT**	**1.00**	**1.00**	**1.00**	**0.96**	**0.96**	**0.94**

Q: Under the premise of a fixed prompt template, can different CoT variants improve performance?

The design goal of ToT/GoT is 'to optimize the search method of the answer space based on existing reasoning templates'
But they cannot change the fundamental problem of whether the 'template is effective' — if the template autonomously generated by the model is suboptimal (such as misjudging the push sequence in stack operations), even increasing multi-path exploration will only waste computational resources in the wrong direction of reasoning, and may even reduce performance.

A: It can be slightly optimized, but cannot replace the core role of the optimal prompt template.

Thanks