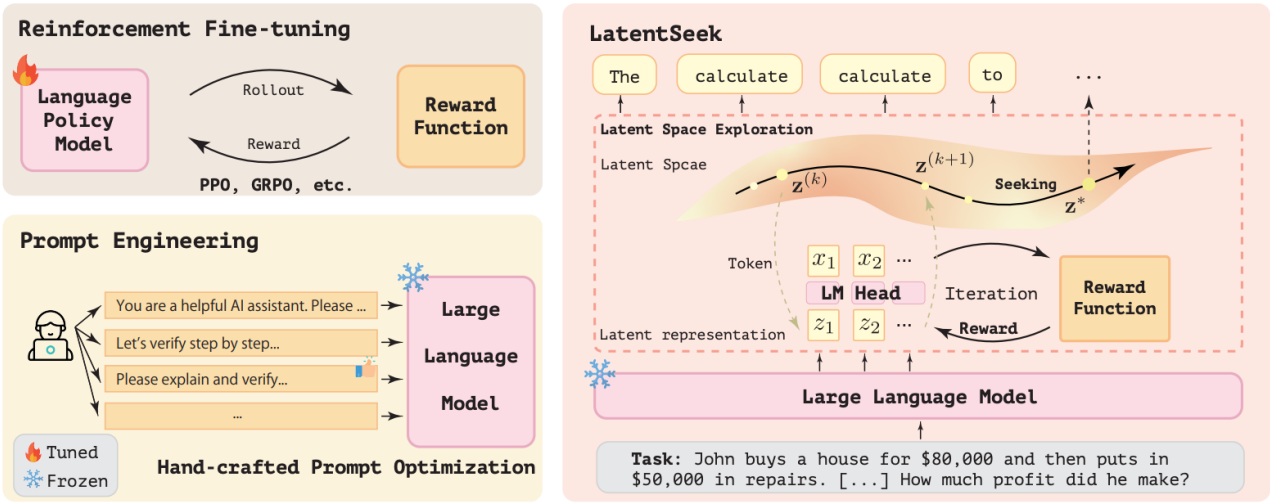
- [x] (2025.5) Hybrid Latent Reasoning via Reinforcement Learning
- [x] (2025.5) Reinforced Latent Reasoning for LLM-based Recommendation
- [x] (2025.5) Seek in the Dark: Reasoning via Test-Time Instance-Level Policy Gradient in Latent Space
- [ ] (2025.5) Think Silently, Think Fast: Dynamic Latent Compression of LLM Reasoning Chains
- [ ] (2025.7) CTRLS: Chain-of-Thought Reasoning via Latent State-Transition
- [x] (2025.5) Continuous Chain of Thought Enables Parallel Exploration and Reasoning
- [ ] (2025.5) Do language models use their depth efficiently?
- [ ] Understanding and minimising outlier features in transformer training

Seek in the Dark: Reasoning via Test-Time Instance-Level Policy Gradient in Latent Space

2025.5.19 BIGAI, Zilong

- **LATENTSEEK** leverages policy gradient to iteratively update latent representations, guided by self-generated reward signal
- Test-Time Instance-Level Adaptation (TTIA) does **not** require parameter updating and operates on a *per-instance basis* during the *testing phase*



Test-Time Instance-Level Adaptation (TTIA)

problem formulation

auto-regressive language model π, a reasoning problem instance c as context prompt:

$$\pi(x|c) = \prod_{t=1}^{T} \pi(x_t|x_{<t}, c)$$

at test time, the ground truth is unknown, introduce a reward function $R(x, c)$:

$$x^* = \text{argmax}_x R(x, c)$$

TTIA with policy gradient in latent space

for a given input sequence x, denote a corresponding sequence of latent representations $z = (z_1, z_2, \cdots, z_T)$, where $z_t \in \mathbb{R}^d$ lies in the latent space of x_t, 这里说的是LM head之前的最后一层hidden state。

则TTIA in the latent space的优化目标如下：

$$z^* = \argmax_z \mathbb{E}_{x\sim\pi(x|z)}[R(x, c)]$$

> comment: 这太奇怪了，这样language model只是encoder的作用？reasoning指望在这最后一层latent通过self modification完成？

- sampling strategy
  - 像本来的模型一样，某个位置的latent只管那个位置的token
  - $\pi(x|z) = \prod_{t=1}^{T} \pi(x_t|z_t)$
- optimization procedure
  - $z \leftarrow z + \eta\nabla_z \square(z)$
  - $\nabla_z \square(z) = \mathbb{E}_{x\sim\pi(x|z)}[R(x, c)\nabla_z \log \pi(x|z)]$
- reward function
  - latent space → greedy decoding → self-reward prompt + pretrained language model → compute the reward
  - $R(x, c) \sim \pi(\cdot|\tilde{x}, c, \text{prompt}_{\text{self-reward}})$

总体思想：用LLM的self-reward当reward function来优化最后一层的hidden state，像一个较为华丽的self-verification+self-consistency

performance

应该保持BoN和LATENTSEEK iteration数相同的：

Methods \ Model	Qwen2 7B	Qwen2.5 1.5B	Qwen2.5 7B	Qwen2.5 14B	LLaMA3.1 8B	Mistral 7B	Avg
GSM8K: Prompt 1							
CoT	68.01	68.08	88.48	92.03	50.19	23.28	65.01
BoN ($N = 3$)	79.76	68.31	89.08	92.27	72.93	24.11	71.30
SFT†	65.86	49.20	72.55	82.39	40.33	24.92	55.88
SFT (Magpie 25K)†	76.50	66.48	83.01	90.30	70.81	13.72	66.80
iCoT† (Deng et al., 2024)	47.54	23.28	41.02	-	47.08	**49.13**	41.61
LATENTSEEK (*Self*)	**84.38**$_{+16.37}$	**69.37**$_{+1.29}$	**89.46**$_{+0.98}$	**92.49**$_{+0.46}$	**78.54**$_{+28.35}$	23.88$_{+0.60}$	**73.02**$_{+8.01}$
LATENTSEEK (*PSRM*)	92.80$_{+24.79}$	85.44$_{+17.36}$	93.93$_{+5.45}$	95.91$_{+3.88}$	88.55$_{+38.36}$	65.96$_{+42.68}$	87.10$_{+22.09}$
GSM8K: Prompt 2							
CoT	65.20	15.31	66.41	91.81	69.07	41.70	58.25
BoN ($N = 3$)	61.33	8.49	74.04	92.27	75.97	43.06	59.19
LATENTSEEK (*Self*)	**80.21**$_{+15.01}$	**44.20**$_{+28.89}$	**85.06**$_{+18.65}$	**92.72**$_{+0.91}$	**83.70**$_{+14.63}$	**44.58**$_{+2.88}$	**71.74**$_{+13.49}$
LATENTSEEK (*PSRM*)	92.80$_{+27.60}$	67.48$_{+52.17}$	93.78$_{+27.37}$	96.13$_{+4.32}$	94.77$_{+25.70}$	78.24$_{+36.54}$	87.20$_{+28.95}$
MATH-500: Prompt 1							
CoT	51.40	54.80	72.80	77.20	47.60	16.40	53.37
BoN ($N = 3$)	53.40	47.40	75.40	78.80	51.20	16.60	53.80
SFT (Magpie 25K)†	46.6	44.40	55.40	68.2	31.00	3.2	41.46
LATENTSEEK (*Self*)	**57.40**$_{+6.00}$	**55.60**$_{+1.80}$	**75.60**$_{+2.80}$	**80.00**$_{+2.80}$	**54.60**$_{+7.00}$	**17.20**$_{+0.60}$	**56.73**$_{+3.36}$
LATENTSEEK (*PSRM*)	75.80$_{+24.40}$	75.80$_{+21.00}$	86.40$_{+13.60}$	87.20$_{+10.00}$	74.60$_{+27.00}$	41.80$_{+25.40}$	73.60$_{+20.23}$
MATH-500: Prompt 2							
CoT	37.40	27.60	53.80	68.00	40.40	8.20	39.23
BoN ($N = 3$)	41.60	24.80	55.80	64.20	44.40	**11.60**	40.40
LATENTSEEK (*Self*)	**44.80**$_{+7.39}$	**32.20**$_{+4.60}$	**57.60**$_{+3.80}$	**71.00**$_{+3.00}$	**47.00**$_{+6.60}$	9.80$_{+1.60}$	**43.73**$_{+4.50}$
LATENTSEEK (*PSRM*)	67.80$_{+30.40}$	57.60$_{+30.00}$	77.20$_{+23.40}$	86.00$_{+18.00}$	66.40$_{+26.00}$	31.40$_{+23.20}$	64.39$_{+25.16}$
AIME2024: Prompt1							
CoT	0.0	3.33	6.67	10.00	0.00	0.00	3.33
BoN ($N = 3$)	0.00	0.00	10.00	**16.67**	0.00	0.00	4.45
SFT (Magpie 25K)†	3.33	0.0	3.33	10.00	3.33	0.00	3.33
LATENTSEEK (*Self*)	**3.33**$_{+3.33}$	**6.67**$_{+3.33}$	**13.33**$_{+6.37}$	**16.67**$_{+6.67}$	**10.00**$_{+10.00}$	**3.33**$_{+3.33}$	**8.33**$_{+4.45}$
LATENTSEEK (*PSRM*)	13.33$_{+13.33}$	6.67$_{+3.33}$	16.67$_{+10.00}$	26.67$_{+16.67}$	10.00$_{+10.00}$	3.33$_{+3.33}$	12.78$_{+9.45}$
AIME2024: Prompt2							
CoT	0.00	0.00	0.00	3.33	0.00	**0.00**	0.56
BoN ($N = 3$)	**3.33**	0.00	6.67	**10.00**	**6.67**	**0.00**	4.45
LATENTSEEK (*Self*)	**3.33**$_{+3.3}$	**3.33**$_{+3.3}$	**13.33**$_{+13.33}$	**10.00**$_{+6.67}$	**6.67**$_{+6.67}$	**0.00**$_{+0.00}$	**5.55**$_{+5.00}$
LATENTSEEK (*PSRM*)	3.33$_{+3.33}$	6.67$_{+6.67}$	13.33$_{+13.33}$	23.33$_{+20.00}$	10.00$_{+10.00}$	0.00$_{+0.00}$	9.44$_{+8.88}$

Hybrid Latent Reasoning via Reinforcement Learning

2025.5.24 UIUC

- introduce hybrid reasoning policy optimization (HRPO), an RL-based hybrid latent reasoning approach that
  - (1) integrates prior hidden states into sampled tokens with a learnable gating mechanism
  - (2) initializes training with predominantly token embeddings while progressively incorporating more hidden features

Methods

- 把最后一层hidden states对token embedding做加权和token embedding加起来，然后做正常的RL。

- 有一个gating机制决定hidden state和sampled token's token embedding的占比

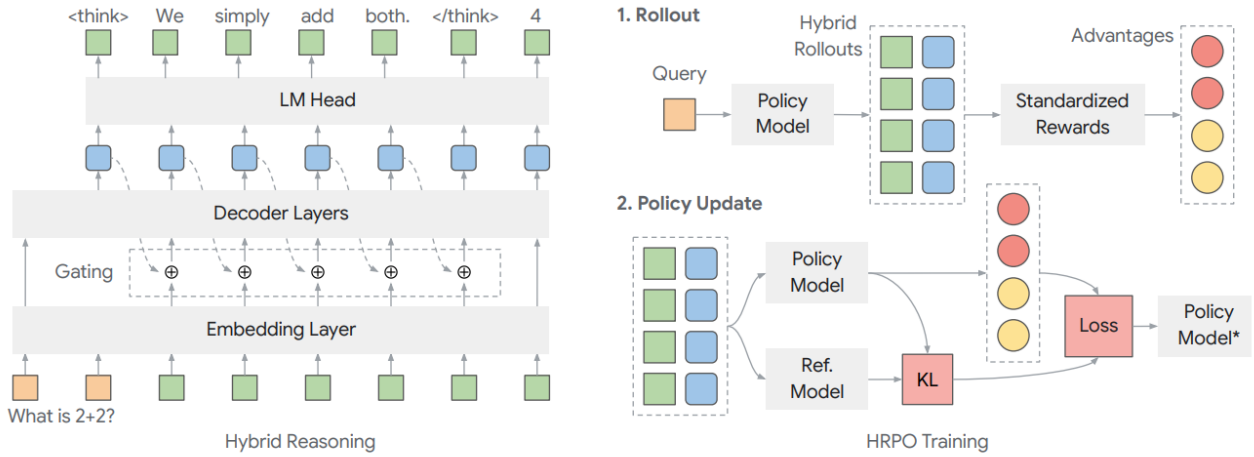


Figure 2: Hybrid reasoning with gating (left) and hybrid reasoning policy optimization (right). During rollouts, the reasoning trajectory is generated hybridly with both discrete tokens and latent features, and for policy update, we compute the HRPO loss using the hybrid rollout buffer to update the model.

Performance

	NQ	TriviaQA	HotpotQA	2WikiMQA	Bamboogle	Average
Qwen2.5-7B-Instruct						
QA	0.134	0.408	0.183	**0.250**	0.120	0.219
CoT	0.048	0.185	0.092	0.111	0.232	0.134
IRCoT	0.224	0.478	0.133	0.149	0.224	0.242
Search-o1	0.151	0.443	0.187	0.176	**0.296**	0.251
RAG	**0.349**	**0.585**	**0.299**	0.235	0.208	**0.335**
Qwen2.5-1.5B-Instruct						
SFT	0.094	0.193	0.129	0.210	0.024	0.130
RAG	0.288	0.477	0.228	0.203	0.072	0.254
PPO	0.327	0.527	0.256	0.242	0.184	0.307
GRPO	0.293	0.480	0.202	0.213	0.120	0.261
HRPO (Ours)	**0.364**	**0.553**	**0.273**	**0.276**	**0.216**	**0.337**
Qwen2.5-3B-Instruct						
SFT	0.249	0.292	0.186	0.248	0.112	0.217
RAG	0.348	0.544	0.255	0.226	0.080	0.291
PPO	0.356	0.563	0.304	0.293	0.240	0.351
GRPO	**0.381**	0.570	0.308	0.303	0.272	0.367
HRPO (Ours)	0.378	**0.593**	**0.316**	**0.318**	**0.296**	**0.380**

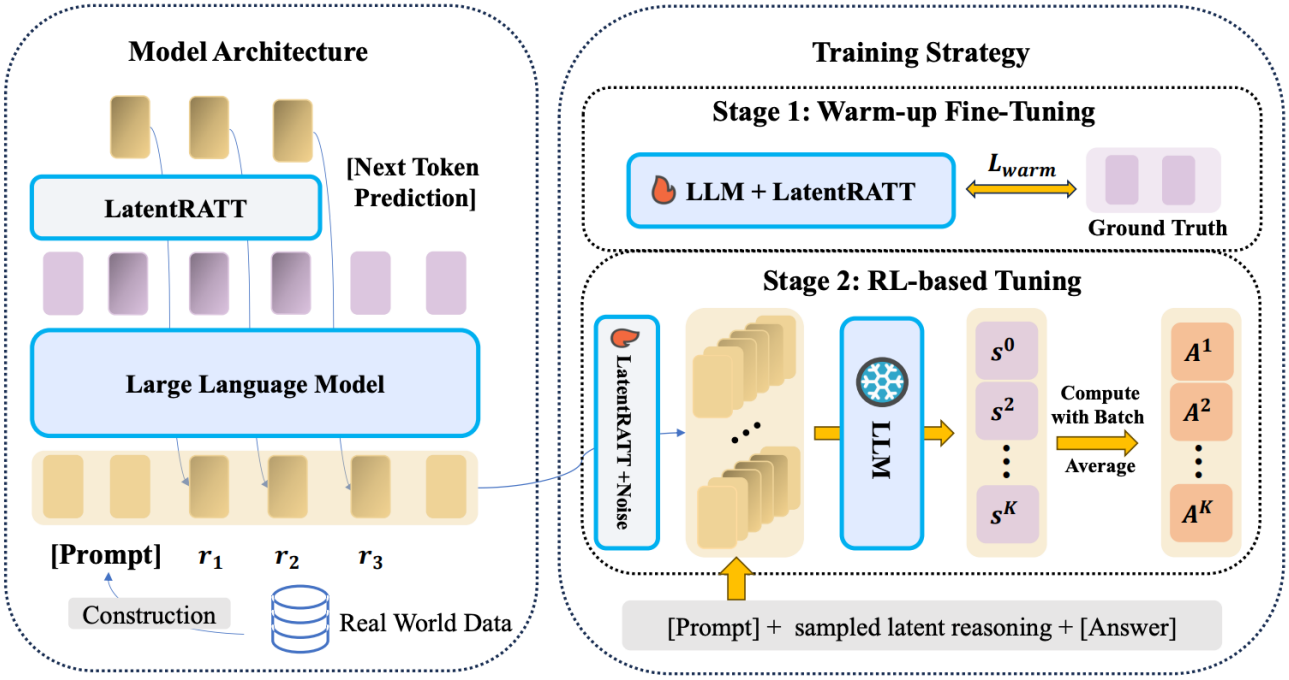
Reinforced Latent Reasoning for LLM-based Recommendation

- discuss applications in reconmmendation system
- two-stage strategy:

    - first, supervised finetuning to initialize the latent reasoning module
    - second, pure RL training to encourage exploration through a rule-based reward design
- add an attention layer on top of the LLM's final decoding layer ⟶ extracts information from the final hidden states of the LLM to generate a latent reasoning token aligned with the LLM's input embedding space

> 为什么是一个attention层?

Latent reasoning architecture



1. introduce an additional attention layer on top of the final decoding layer to explicitly generate latent thought tokens
    1. dedicated reasoning generator that aggregates contextual information?
    2. ensure the generated tokens are aligned with the LLM's input embedding space
2. 定长latent token

⭐ Continuous Chain of Thought Enables Parallel Exploration and Reasoning

- continuously-valued tokens (CoT2)
    - examines the benefits of CoT2 through logical reasoning tasks that inherently require search capabilities
    - provide optimization and exploration methods for CoT2
- interpretability
    - CoT2 allows the model to track multiple traces in parallel and quantify its benefits for inference efficiency
    - one layer transformer equipped with CoT2 can provably solve the combinatorial "subset sum problem"

- several sampling strategies:
    1. samples and composes K discrete tokens ar each decoding step to control the level of parallelism (reduce to standard CoT when $\mathrm{K} = 1$)
    2. continuous exploration over the probability simplex
    3. policy optimization with CoT2 indeed inproves the performance of the model beyond its initial discrete and continuous supervision

Introduction

Motivation:

1. modern language models may not fully utilize their potential for a few reasons:
    1. discrete sampling of tokens limits the model to emitting at most $\log_2(\mathrm{v})$ (v is the size of vocabulary) bits per sample, or more specifically, the Shannon entropy of the softmax output. This contrasts with the $\mathrm{O}(\mathrm{d})$ bits each token embedding can store
    2. Secondly, discrete sampling can cause the model to commit to certain solutions and avoid exploring alternatives

Method: CoT with Continuous Tokens (CoT2), built on COCONUT

- rather than the model sampling a single token from the vocabulary, it samples or deterministically *selects a continuous superposition of tokens according to the softmax output*

Contributions:

- Mechanistic and theoretical study of CoT2
    - theoretical: single-layer transformer can solve Minimum Non-Negative Sum (MNNS) using CoT2
    - theoretical study for:
        - **Base CoT2**: deterministic inference which creates and feeds continuous tokens using full softmax output at each step
        - **CoT2-MTS (multi-token sampling)**: samples K discrete tokens from softmax and averages them to form a continuous token (maybe like top-k version of Base CoT2)
- Supervision and reinforcement for CoT2
    - distill, like distill multiple CoT traces into CoT2 continuous tokens
    - policy optimization

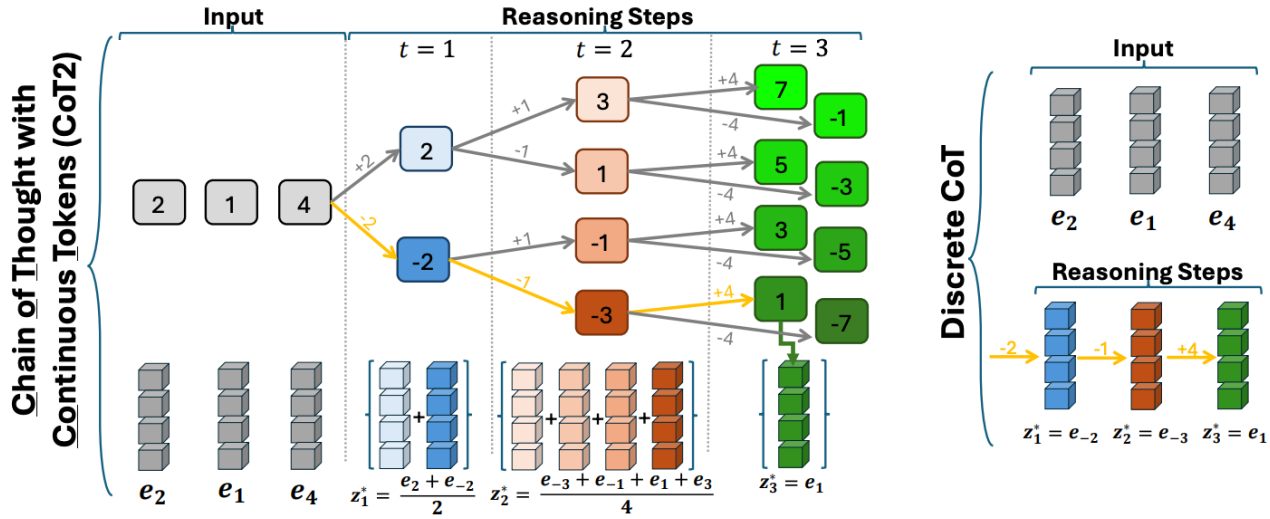


Figure 1: Illustration of CoT2 and discrete CoT for Minimum Non-Negative Sum (MNNS) task with $m = 3$. The input numbers are $2, 1, 4$, and the correct path for this task $(-2, -3, 1)$ is highlighted with yellow arrows and corresponds to the discrete CoT supervision. CoT2 supervision for the reasoning steps $t \in \{1, \ldots, m-1\}$ is the average of embeddings of reachable states, and for $t = m$ is the embedding corresponding to the answer.

Problem Setup

Objective: given $X \in \mathbb{R}^{n \times d}$, where each of the n rows is a d-dimensional embedding vector, the objective is to output m tokens where the m^{th} token to be the final answer.

除了final answer，前$m-1$个token均为continuous： For the first $m-1$ steps, the model outputs continuous tokens $z_{t\,t\in[m-1]}$, at the final step $t = m$, the model outputs a discrete token z_m from a vocabulary of size v.

对于$1 \le t \le m-1$, the model outputs the following probability distribution over v vocabulary entries via a softmax operation:

$$LM_\theta(\cdot|z_{<t}, X) := \alpha_t$$

$$z_t = E^\top \alpha_t \in \mathbb{R}^d$$

where E is the embedding matrix $E = [e_1, \cdots, e_v]^\top \in \mathbb{R}^{v \times d}$

CSFT: A Supervised Training Method for CoT2

supervise at each token, specify a target probability distribution: $$ \alpha_t^* = [\alpha_{t,1}^*, |cdots, |alpha_{t, v}^*] $$

train the model to minimize the cross entropy between model output and this distribution

2 ways to provide prefix:

1. use ground truth prefix
2. use previous generated continuous tokens

latent token数目固定

Reinforcement Learning Methods for CoT2

Algorithm 1 Multi-Token Sampling GRPO for Continuous Token Generation

Input: Initial policy $\mathrm{LM}_{\theta_{\text{init}}}$; hyperparameters K, G, m, ϵ, β.

$\mathrm{LM}_{\theta}, \mathrm{LM}_{\theta_{\text{ref}}} \leftarrow \mathrm{LM}_{\theta_{\text{init}}}$
for iteration = 1, 2, . . . , I and **for** step = 1, 2, . . . , S **do**
  Sample a batch of inputs $\{\boldsymbol{X}^{(b)}\}_{b=1}^{B}$
  Update $\mathrm{LM}_{\theta_{\text{old}}} \leftarrow \mathrm{LM}_{\theta}$
  for each input $\boldsymbol{X}$ in the batch and **for** each trajectory $i = 1, \ldots, G$ from that $\boldsymbol{X}$ **do**
    for each token step $t = 1, \ldots, m$ **do**
      if $t < m$ **then** ▷ Continuous token
        Sample K tokens $\{\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_K}\}$ from $\boldsymbol{\alpha}_t^{(i),\text{old}}$ to create continuous token $\boldsymbol{z}_t \leftarrow \frac{1}{K}\sum_{r=1}^{K} \boldsymbol{e}_{i_r}$.
        Policy ratio for continuous token $r_t(\theta) \leftarrow \left(\prod_{r=1}^{K} \alpha_{t,i_r}^{(i)} / \prod_{r=1}^{K} \alpha_{t,i_r}^{(i),\text{old}}\right)^{\frac{1}{K}}$.
      else ▷ Final discrete token
        Sample $\boldsymbol{z}_m = \boldsymbol{e}_j$ from $\boldsymbol{\alpha}_m^{(i),\text{old}}$.
        Policy ratio for discrete token $r_m(\theta) \leftarrow \alpha_{m,j}^{(i)} / \alpha_{m,j}^{(i),\text{old}}$.
    Obtain advantage estimates $\hat{A}_{i,t}$ for each token t in each trajectory $\boldsymbol{Z}^{(i)}$ and calculate objective.
  Update θ to minimize $\mathcal{L}_{\text{GRPO}}(\theta)$.

Output LM_{θ}

Think Silently, Think Fast: Dynamic Latent Compression of LLM Reasoning Chains

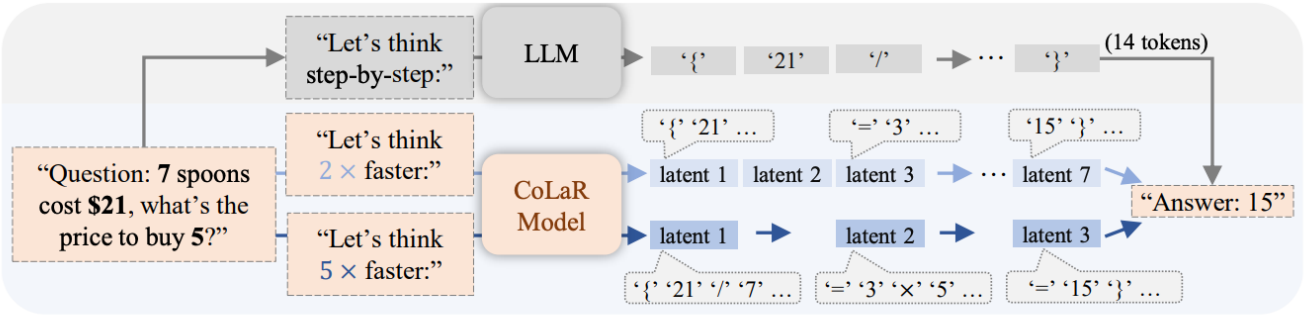


Figure 1: Our proposed Compressed Latent Reasoning Model (CoLaR) performs dynamic-speed reasoning by auto-regressively predicting latent variables, each compressing information from multiple word tokens. Simply prompting to reason faster enables CoLaR to predict more informative latents.

overview

- 2-stage training approach
  - SFT: next-token prediction by incorporating an auxiliary next compressed embedding prediction objective
  - RL
- perform reasoning at a dense latent level; dynamically adjust reasoning speed

motivation regarding previous attempts

1. regarding efficiency work about long-CoT
   1. enhancing efficiency at the token level, primarily by identifying and skipping less informative tokens
   2. prompting models to generate more concise reasoning steps

3. dynamically terminating reasoning when the model exhibits high confidence in a trial answer

2. regarding latent CoT
    1. Initial efforts attempt to “internalize” reasoning knowledge by curriculum learning or knowledge distillation
    2. Some works focus on the potential inside LLMs by looping or skipping some intermediate LLM layers
    3. Recent innovations have introduced auto-regressive prediction of latent representations for efficient reasoning

```
limitations:
1. fixed-length reasoning chains
2. use deterministic latent reasoning processes
```

method

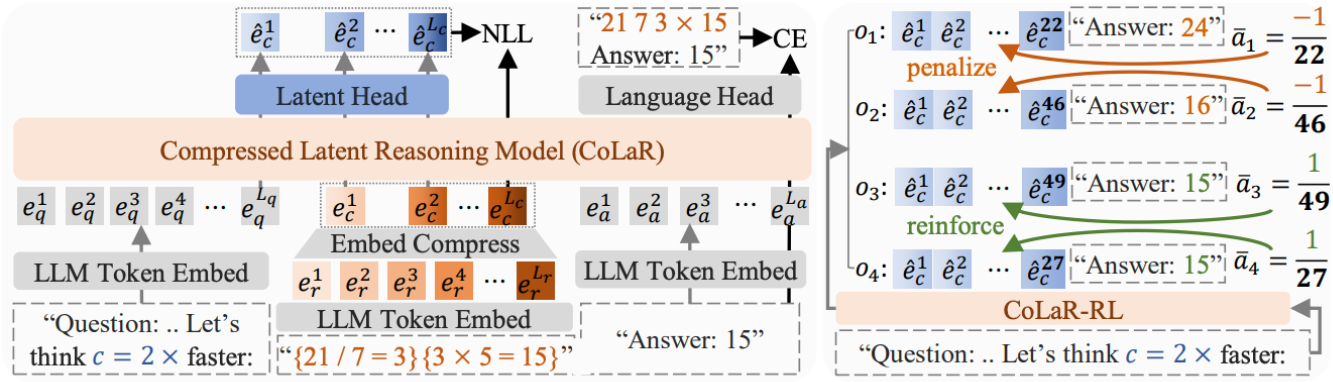


Figure 2: Our proposed method CoLaR consisting an LLM backbone and a Latent Head. During the **SFT stage (left)**, for each training step, CoLaR first compresses embeddings $\mathbf{e}_r$ of the original reasoning chain into compressed embeddings $\mathbf{e}_c$ with a compression factor c randomly selected from the range $[1, c_{max}]$. Then, CoLaR is trained to predict: i) the compressed reasoning embeddings via the Latent Head, and ii) the compressed reasoning tokens and answer tokens through the Language Head. During the **RL stage (right)**, for every question input, CoLaR samples a group of G outputs $o_{1:G}$ consisting of the latent reasoning chain and the predicted answer. We then calculate the relative rewards $a_{1:G}$ for each output, and the rewards are averaged on each token ($\bar{a}_i$), encouraging CoLaR to explore diverse latent reasoning pathways and exploit those more compact ones.

- utilize an auxiliary next compressed embedding prediction task in SFT stage
    - at each training step, CoLaR first samples a random compression factor $c \in [1, c_{max}]$ and merges the embeddings of c

1. Reasoning token compression and understanding
    1. 根据compression rate，begin each training step by randomly sampling $c \in [1, r_{max}]$, for every r consecutive reasoning token embeddings $e_r^{k:k+r}$, the embedding compress module generates a compressed embedding e_c^k
    2. embedding compress module:
        1. apply mean pooling:
            - Simply averaging these embeddings can distort the original distribution.
            - For instance, consider two uncorrelated distributions $A \sim N(\mu, \sigma^2)$, $B \sim N(\mu, \sigma^2)$

- mean pooling would alter the original distribution to $\frac{A+B}{2}$, effectively scaling the variance by $\frac{1}{2}$. We found that, for most pre-trained LLMs, the distributions of embeddings are centered at $\mu \approx 0$. Thus, to prevent distortion of the original embedding distribution of LLMs, the Embedding Compress module only scales the sum of the r embeddings by $\frac{1}{\sqrt{r}}$

3. 怎么给loss
    1. 只通过answer来给不佳，监督太稀疏了没法收敛
    2. train一个model来predict compressed reasoning token, for each compressed token embedding, CoLaR should be able to **read** and **predict** tokens in groups of *r*. $$ \mathcal{L}*{|text{comp}} = -|frac{1}{L_a+L_c}|sum|limits*{i=1}^{L_a+L_c} \log p([\textbf{t}_c, \textbf{t}_a]^i) $$

2. Next compressed embedding prediction

    1. explore in the latent space ⟶ given the current hidden states h_c^i output by □, the latent head □ predicts both the mean μ_c^{i+1} and the standard deviation σ_c^{i+1} of the next embedding's distribution ⟶ $e_c^{i+1} = \mu_c^{i+1} + \sigma_c^{i+1}\epsilon$

    2. the latent head is primarily trained using the negative log-likelihood loss, for a prediction at position i, this can be formulated as:

$$□_{\text{latent}}(i) = -\log p(e_c^i | \mu_c^i, \sigma_c^i) = \frac{(e_c^i - \mu_c^i)^2}{2\sigma_c^i} + \log \sigma_c^i$$

    3. we empirically found that CoLaR with NLL loss tends to under-fit on simpler math reasoning datasets that require less exploratio ⟶ propose a soft-MSE loss that combines Mean Squared Error with an entropy regularization term $$ \mathcal{L}*{|text{latent}}(i) = |mathbb{E}*\epsilon [(\mu_c^i+\sigma_c^i \epsilon -e_c^i)^2] - \alpha \left(\frac{1}{2}\log (2\pi e (\sigma_c^i)^2)\right) $$

3. Exploration with reinforcement learning

    1. use GRPO

Performance

- use Llama-3.2-1B

Table 1: Experiment results of baseline methods and CoLaR on four grade-school math reasoning datasets. We test the methods for five times with different random seeds to report the averaged number and 95% confidence interval (±) on accuracy (Acc. %) and reasoning chain length (# L). CoLaR-c denotes a same CoLaR model tested with different compression factors c. For ablation methods (marked in gray), suffixes DL, OC, MP and NLL denote CoLaR with a Deterministic Latent head, training withOut Compressed reasoning chain in cross entropy labels, using Mean Pooling to compress embeddings, and training with NLL loss, respectively.

	GSM8k-Aug		GSM-Hard		SVAMP		MultiArith		Average	
	Acc.	# L	Acc.	# L	Acc.	# L	Acc.	# L	Acc.	# L
CoT	$49.4_{\pm.72}$	$25.6_{\pm.11}$	$11.9_{\pm.16}$	$34.2_{\pm.11}$	$59.8_{\pm.29}$	$12.1_{\pm.03}$	$93.2_{\pm.49}$	$13.7_{\pm.09}$	53.6	21.4
iCoT	$19.8_{\pm.23}$	$0.00_{\pm.00}$	$3.87_{\pm.16}$	$0.00_{\pm.00}$	$36.4_{\pm.51}$	$0.00_{\pm.00}$	$38.2_{\pm.66}$	$0.00_{\pm.00}$	24.6	0.00
Coconut	$23.1_{\pm.28}$	$6.00_{\pm.00}$	$5.49_{\pm.33}$	$6.00_{\pm.00}$	$40.7_{\pm.65}$	$6.00_{\pm.00}$	$41.1_{\pm.24}$	$6.00_{\pm.00}$	27.6	6.00
Distill	$13.3_{\pm.62}$	$6.00_{\pm.00}$	$2.97_{\pm.24}$	$6.00_{\pm.00}$	$21.7_{\pm.73}$	$6.00_{\pm.00}$	$19.2_{\pm.83}$	$6.00_{\pm.00}$	14.3	6.00
CoLaR-5	$26.8_{\pm.17}$	$5.57_{\pm.02}$	$5.87_{\pm.10}$	$6.53_{\pm.01}$	$48.4_{\pm.45}$	$2.95_{\pm.02}$	$86.4_{\pm.35}$	$3.21_{\pm.01}$	41.7	4.57
- DL	$26.7_{\pm.11}$	$5.74_{\pm.01}$	$5.53_{\pm.11}$	$8.20_{\pm.04}$	$48.3_{\pm.05}$	$2.90_{\pm.01}$	$84.5_{\pm.19}$	$3.22_{\pm.01}$	41.3	5.02
- OC	$24.8_{\pm.27}$	$5.14_{\pm.12}$	$6.46_{\pm.11}$	$5.49_{\pm.06}$	$46.5_{\pm.18}$	$2.85_{\pm.01}$	$85.9_{\pm.22}$	$3.13_{\pm.01}$	40.1	4.15
- MP	$20.6_{\pm.22}$	$5.61_{\pm.02}$	$4.20_{\pm.07}$	$6.18_{\pm.02}$	$47.7_{\pm.41}$	$2.96_{\pm.01}$	$80.7_{\pm.59}$	$3.20_{\pm.01}$	38.3	4.49
- NLL	$20.3_{\pm.64}$	$5.99_{\pm.06}$	$4.52_{\pm.39}$	$16.6_{\pm.25}$	$43.9_{\pm.43}$	$3.06_{\pm.03}$	$81.6_{\pm.23}$	$3.20_{\pm.02}$	37.6	8.01
CoLaR-2	$40.1_{\pm.20}$	$12.7_{\pm.02}$	$9.08_{\pm.03}$	$14.0_{\pm.07}$	$54.9_{\pm.20}$	$6.11_{\pm.01}$	$91.3_{\pm.12}$	$7.35_{\pm.01}$	48.8	10.0
- DL	$39.7_{\pm.18}$	$12.8_{\pm.01}$	$8.84_{\pm.06}$	$17.2_{\pm.09}$	$54.3_{\pm.23}$	$6.10_{\pm.01}$	$90.1_{\pm.17}$	$7.46_{\pm.01}$	48.2	10.9
- OC	$39.1_{\pm.33}$	$12.3_{\pm.04}$	$8.96_{\pm.01}$	$16.9_{\pm.13}$	$54.7_{\pm.18}$	$6.08_{\pm.02}$	$90.1_{\pm.25}$	$7.36_{\pm.01}$	48.2	10.6
- MP	$36.9_{\pm.30}$	$12.4_{\pm.02}$	$8.46_{\pm.19}$	$12.0_{\pm.05}$	$54.1_{\pm.42}$	$6.14_{\pm.01}$	$86.8_{\pm.20}$	$7.43_{\pm.01}$	46.6	9.49
- NLL	$32.3_{\pm.51}$	$12.2_{\pm.04}$	$7.57_{\pm.16}$	$16.6_{\pm.25}$	$51.0_{\pm.24}$	$5.50_{\pm.03}$	$88.3_{\pm.41}$	$7.09_{\pm.02}$	44.8	10.3

Table 2: Experimental results on the challenging MATH dataset. We evaluate our proposed method CoLaR on two base models and three settings: -DL denotes using a Deterministic Latent head, -NLL denotes CoLaR trained with NLL Loss as $\mathcal{L}_{\text{latent}}$, which is our main method, and - /w GRPO denotes the post-trained CoLaR-NLL with GRPO reinforcement learning process. We calculate the performance gain between CoLaR-NLL and CoLaR-NLL-RL to highlight the effectiveness of reinforcement learning. Compression factor c and # L_{max} are set to 2 and 128, respectively.

	DeepSeek-R1-Distill-Qwen-1.5B		Llama-3.2-1B-Instruct	
	Acc.	# L	Acc.	# L
CoT	$23.5_{\pm.29}$	$209_{\pm1.6}$	$9.71_{\pm.33}$	$210_{\pm1.4}$
CoLaR-DL	$9.04_{\pm.12}$	$99.4_{\pm.25}$	$3.07_{\pm.28}$	$134_{\pm.46}$
CoLaR-NLL	$8.94_{\pm.21}$	$56.8_{\pm.14}$	$5.28_{\pm.16}$	$83.1_{\pm.52}$
CoLaR-NLL-RL	$14.3_{\pm.25}$ (5.36% ↑)	$9.79_{\pm.40}$ (82.8% ↓)	$7.08_{\pm.07}$ (1.80% ↑)	$16.1_{\pm.14}$ (80.6% ↓)
- w/o average	$13.8_{\pm.14}$	$128_{\pm.00}$	$0.00_{\pm.00}$	$128.0_{\pm.00}$

Do language models use their depth efficiently?

Motivation

Modern LLMs are increasingly deep, and depth correlates with performance, albeit with diminishing returns. However, do these models use their depth efficiently? Do they compose more features to create higher-order computations that are impossible in shallow models, or do they merely spread the same kinds of computation out over more layers?

- analyze the residual stream of Llama-3.1 & Qwen 3 family of models
- results:
    1. compare the output of the sublayers to the residual stream reveals that layers in the second half contribute much less than those in the first half, **with a clear phase transition**
    2. skip layers in the second half has a much smaller effect on future computations and output predictions

3. for multi-hop tasks, we are unable to find evidence that models are using increased depth to compose subresults in examples involving many hops
4. we seek to directly address whether deeper models are using their additional layers to perform new kinds of computation
    1. train linear maps from the residual stream of a shallow model to a deeper one. We find that layers with the same relative depth map best to each other, suggesting that the **larger model simply spreads the same computations out over its many layers**.

All this evidence suggests that deeper models are **not** using their depth to learn new kinds of computation, but only **using the greater depth to perform more fine-grained adjustments to the residual**.

more fine-grained motivation

- 一方面，之前很多工作发现，对layer做pruning不会损伤模型性能；实验上，d Gromov et al. were able to remove half of the layers from the network without significantly affecting performance on MMLU (but not for math)
- 另一方面，可解释性研究发现模型不同层确实在做不同的操作

本文的核心问题：**do deeper LLMs use their depth to compose more features to create higher-order computations that are impossible in shallow models, or do they merely spread the same kinds of computation out over more layers?**

All the models we analyze are pre-layernorm Transformers [6, 17]. A Transformer layer l is constructed as follows:

$$\boldsymbol{a}_l = \text{SelfAttention}_l(\text{Norm}(\boldsymbol{h}_l)) \tag{1}$$

$$\hat{\boldsymbol{h}}_l = \boldsymbol{h}_l + \boldsymbol{a}_l \tag{2}$$

$$\boldsymbol{m}_l = \text{MLP}_l(\text{Norm}(\hat{\boldsymbol{h}}_l)) \tag{3}$$

$$\boldsymbol{h}_{l+1} = \hat{\boldsymbol{h}}_l + \boldsymbol{m}_l \tag{4}$$

Here, $\boldsymbol{h}_l \in \mathbb{R}^{T \times d_{\text{model}}}$ is the residual stream and $\boldsymbol{a}_l, \boldsymbol{m}_l \in \mathbb{R}^{T \times d_{\text{model}}}$ are the outputs of the SelfAttention and MLP layers, respectively, where T is the length of the current input sequence and d_{model} is the width of the residual stream. Norm$(\cdot)$ is some token-wise normalization, traditionally layer normalization [18], but usually replaced with RMSNorm [19]. We call SelfAttention$_l(\cdot)$ and MLP$_l(\cdot)$ *sublayers*.

Methods

- 模型主要用：Llama-3.1-70B
- 可解释性repo：NDIF and NNsight

1. How do the Layers Interact With the Residual Stream?

因为所有Transformer中跟residual stream的互动都是additive的，所以可预期的是，$||h_l||_2$会随layer增长而增长。

- residual growth在outlier feature和universal transformer中都有被观察到过。

在初始时，$||a_l||_2$和$||m_l||_2$在期望意义下时一样的，as input都做了normalization。因此，later layer相比earlier layers对residual的贡献减少：改变residual方向更难。

> 为什么later layer贡献更少？

> 因为norm逐层变大，而$||a_l||_2$和$||m_l||_2$期望不变，则是一个较小的数加到一个较大的数上去，然后再做norm，故影响逐层变小

训练时，模型可以通过在later layer增加weight norm来补偿这种residual贡献减少的趋势。然而，训练时加了weight decay，所以这种compensate又会被抑制。

此处，作者想通过residual growth的角度来看每一层的贡献度。

1. the relative contribution of sub layers.
    - $||h_l||_2, ||a_l||_2$和$||m_l||_2$ 随层的变化如下图：

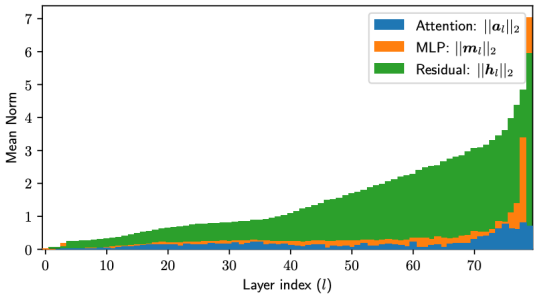
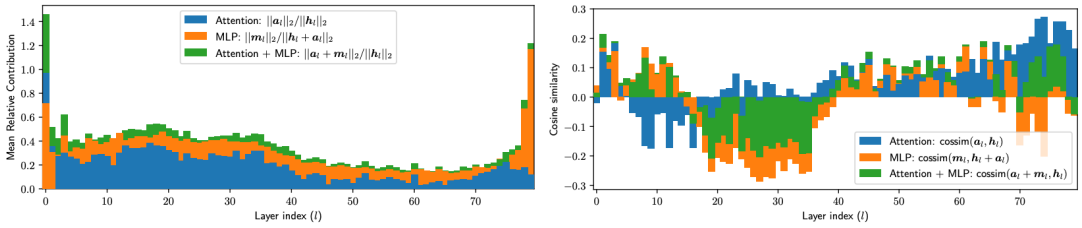


Figure 11: L2 norm of layer inputs and the sublayer contributions before summing into the residual for Llama 3.1 70B. Norm for each layer.

    - relative contribution of each (sub)layer ($\frac{||a_l+m_l||_2}{||h_l||}$, $\frac{||a_l||_2}{||h_l||_2}$ and $\frac{||m_l||_2}{||h_l+a_l||_2}$)
        - 对后面层的影响在中间layer的时候突然下降，last few layers影响又上升



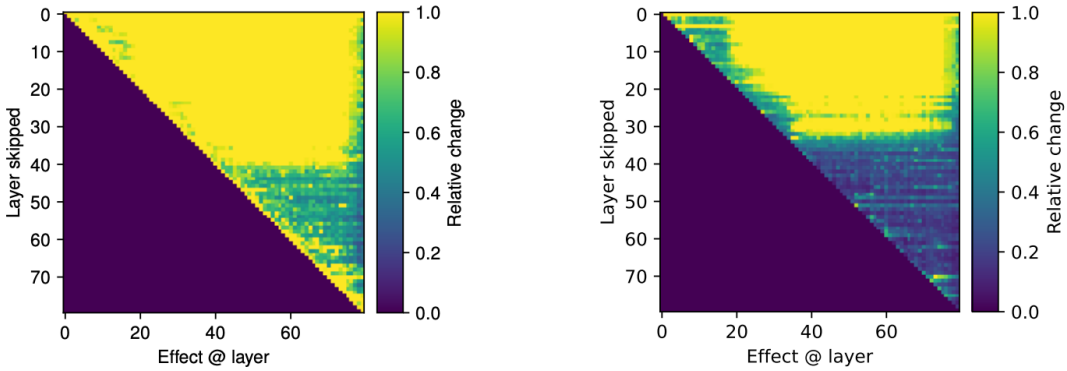
(a) Relative norm of (sub)layer contributions (b) Cossims of (sub)layer contributions and the residual

Figure 2: Influence of layers and sublayers on the residual stream for Llama 3.1 70B. (a) Norm of contributions relative to the residual stream. A sharp drop is visible near the middle; later layers change the residual much less, with the exception of the last few layers. (b) Cosine similarity between the contributions and the corresponding residual shows a phase change at the middle of the network.

2. the cosine similarity between the residual and sublayer contributions
    1. $\text{cossim}(m_l + a_l, h_l)$
    2. Where features are orthogonal to each other, zero cosine similarity corresponds to writing a new feature to the residual, **negative values correspond to erasing features**, and **positive values mean strengthening an existing feature**.
    3. 后面的层和residual保持相似方向，

2. How do the Layers Influence Downstream Computations?

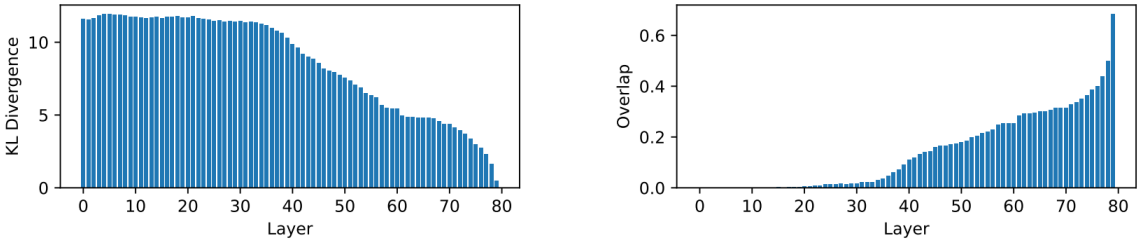
- Causal intervention for measuring the layers' importance for the downstream computation
    - first, we run a prompt through the model and log the residual h_l
    - second, we run the same prompt again, but this time, we skip layer sm by setting $\overline{h}*{s+1}:={h}*{s}, andwelogtheintervenedmodelmodel\overline{h}_l$
    - third, we measure the relative change in the contribution of layer $l>s:\frac{||(h_{l+1}-h_l)-(\overline{h}_{l+1}-\overline{h}*l)||}{||h*{l+1}-h_l||_2}$



(a) Effect of skipping a layer on later layers' contributions in the *all* timesteps.

(b) Effect of skipping a layer on later layers' contributions in *future* timesteps.

- Measuring layer importance for future predictions
  - measure the effect on the future tokens when skipping layers for earlier tokens. We do this by sampling a position $1 < t_s < T - 1$, skipping the layer only for token positions $t \leq t_s$, and measuring the effect only on positions $t > t_s$.
- apply logitlens to the middle-layer hidden states ⟶
  - the prediction refinement seems to start at the same position at the same phase transition as when the layers do not influence the future predictions anymore, where the cosine similarities change sign, and the layer's importance decreases



(a) KL divergence between the model's prediction and Logitlens applied to each layer.

(b) Overlap in top-5 tokens from the model's prediction and Logitlens applied to each layer.

Figure 4: Comparing Logitlens on different layers to the final prediction. (a) KL-divergence. (b) Overlap in the top-5 predicted tokens. Both show that later layers are devoted primarily to refining the output probability distributions, rather than to performing new kind of computation.

3. Do Deeper Problems Use Deeper Computation?

We expect that:

1. transformers use more layers to do deeper computation;
2. later steps of a composite computation should be executed in later layers, so that they can receive the results of earlier subproblems as inputs

but do models really orgnanize their computations this way?

- **Residual erasure interventions**:
  1. integrated gradients: compute the gradient on all answer tokens, but not on the prompt
  2. maximum prediction norm change:
- **The Depth Score**: 1.