

TOWARDS A PROGRESS BAR FOR REASONING: PROGRESS PREDICTION IN LARGE REASONING MOD- ELS

Anonymous authors

Paper under double-blind review

ABSTRACT

Reasoning models that produce long, hidden chains of thought, have emerged as powerful tools for reasoning-intensive and agentic tasks. However, as the time horizons at which these models can operate grow exponentially, it becomes increasingly difficult to know how much progress the model is making on a task, making it challenging for users to set appropriate expectations about completion time. By probing the internal representations of Large Language Models (LLMs), we find evidence that their reasoning progress can be quantified, with simple linear probes achieving 30% accuracy over 10 progress classes and Mean Absolute Error (MAE) of 1.75. Rooted in this insight, we propose a two-stage fine-tuning method that trains existing reasoning models to explicitly generate progress estimates (0-100%) during their reasoning process. We find that the predictions of our best fine-tuned language model for sequences below 16K tokens are on average 10% from the true label.

1 INTRODUCTION

With the advent of Large Reasoning Models (LRMs) (DeepSeek-AI et al., 2025; OpenAI et al., 2024), test-time inference has emerged as a powerful new axis for scaling model performance. This development, however, poses substantial user-interaction challenges: the resources consumed per query can fluctuate significantly, while the model’s reasoning progress often remains opaque or unintelligible. This problem is further compounded by the exponentially growing time horizons at which these models can operate coherently on agentic tasks (Kwa et al., 2025).

While some work has trained models to adhere to fixed predetermined reasoning budgets (Aggarwal & Welleck, 2025; Jaghour et al., 2025; Muennighoff et al., 2025), this strategy fails when budgets cannot be accurately inferred from the question alone, potentially causing models to return incorrect answers despite being capable of solving the problem given a larger reasoning budget.

At the same time, recent research suggests that Large Language Models (LLMs) encode confidence in their hidden states, well in advance of materializing a final answer. For example, (Zhang et al., 2025) uncover that model hidden states contain “look-ahead” information - the ability to predict eventual correctness of a chain-of-thought, before an intermediate answer is fully generated, indicating that LLMs possess an inherent awareness of their reasoning trajectory (Zhang et al., 2025).

Building on this observation, we train probes to study if hidden states encode how far along the model is in completing the reasoning task. We find that a one layer MLP probe gets 30% accuracy on predicting progress over 10 labels, while on average being 1.75 quantiles from the most likely label.

Motivated by this finding, we fine-tune existing reasoning models to intermittently emit progress predictions within dedicated progressbar tags (`<progressbar></progressbar>`) during their reasoning process. We evaluate our approach on reasoning sequences spanning up to 32K tokens and demonstrate that our best fine-tuned model achieves a mean absolute error (MAE) of 10% on progress prediction for sequences up to 16K tokens. However, we find that prediction accuracy degrades with increasing sequence length, highlighting the challenge of maintaining progress estimation quality for extended reasoning chains.

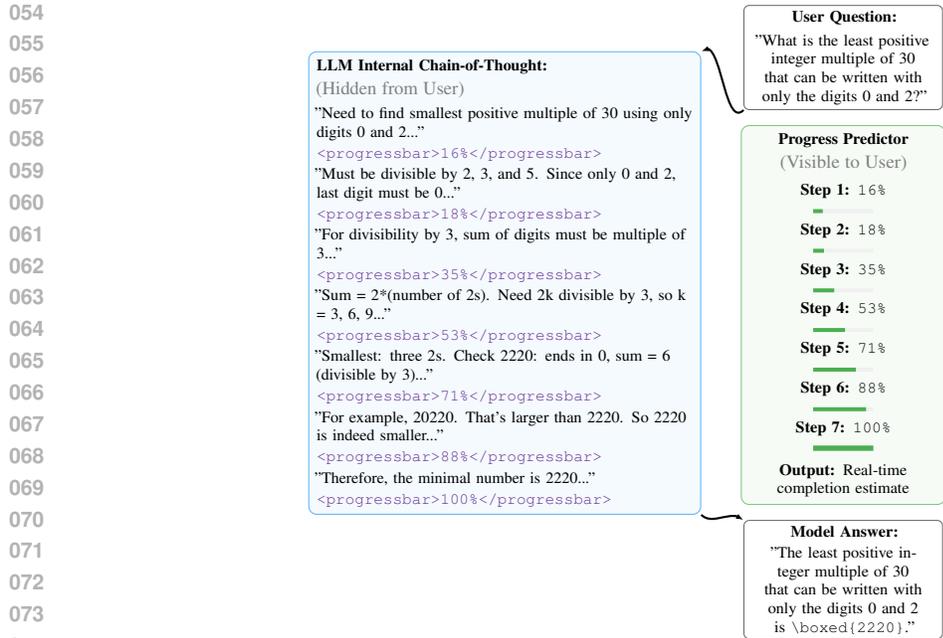


Figure 1: Real-time progress tracking during LLM reasoning. The model intermittently updates the user-facing progressbar by using <progressbar> . . </progressbar>.

Furthermore, our trajectory analysis reveals empirical bounds on progress predictability. By sampling multiple rollouts from prefixes at different positions, we compare progress estimates from sampled completions against the progress calculated using the original complete reasoning sequence, finding MAE to be around 5-10%, suggesting an empirical lower bound on prediction error.

2 PROBLEM DEFINITION AND EXPERIMENTAL SETUP

To the best of our knowledge, progress prediction was first systematically studied in the field of heuristic search algorithms, with Thayer et al. (2012) offering the earliest formal approach.

They define search progress as, given a search algorithm A and a heuristic search problem P , let $E(P)$ denote the total number of nodes expanded by A when solving P . Let $\text{Gen}(P)$ be the nodes already expanded by algorithm A and $\text{Rem}(P, \text{Gen}(P))$ be the number of remaining nodes that will be expanded i.e., $\text{Rem}(P, \text{Gen}(P)) = E(P) - \text{Gen}(P)$.

Definition (Search Progress) The search progress of algorithm A solving problem P after expanding $\text{Gen}(P)$ nodes is then:

$$\text{Prog}(\text{Gen}(P)) = \frac{\text{Gen}(P)}{\text{Gen}(P) + \text{Rem}(P, \text{Gen}(P))} = \frac{\text{Gen}(P)}{E(P)}$$

As $\text{Gen}(P) \leq E(P)$, search progress is thus a value between 0 and 1 and is scale invariant.

This intuition carries well to the domain of reasoning where, we have a problem q and an algorithm A obtained by sampling from the policy, $P_\theta(\cdot | q, \langle \text{think} \rangle)$ parameterized by a reasoning model θ .

Let $\mathbf{C}_k = \{x_1, x_2, \dots, x_k\}$, $x_i \sim P_\theta(\cdot | q, \langle \text{think} \rangle, x_{<i>i</i>})$ be a partial reasoning trace with length \mathbf{K} and let \mathbf{M} be the length of the complete reasoning trace i.e., $|\langle \text{think} \rangle \dots \langle / \text{think} \rangle|$.

1. $\text{Gen}(P) = |\mathbf{C}_k| = \mathbf{K}$,
2. $\text{Rem}(P, \text{Gen}(P))$ is then the expected length over all possible sequences τ starting from \mathbf{C}_k and ending in a terminal state $\langle / \text{think} \rangle$. i.e., $E_{\tau \sim P_\theta(\tau | \mathbf{C}_k)}[|\tau|]$ where $P_\theta(\tau | \mathbf{C}_k)$ is the conditional probability distribution over all such trajectories.

Since exact computation is intractable, we make the approximation $\mathbb{E}_{\tau \sim P_\theta(\tau|C_k)}[|\tau|] \approx \mathbf{m} - \mathbf{k}$, where $|\cdot|$ computes the length.

$$\text{Progress}(C_k) = \frac{\mathbf{k}}{\mathbb{E}_{\tau \sim P_\theta(\tau|C_k)}[|\tau|] + \mathbf{k}} \approx \frac{\mathbf{k}}{\mathbf{m}}$$

Metrics In this paper, we use MAE (mean absolute error) as the main metric for model performance, similar to other work (Huang et al., 2025; Ma et al., 2019; Zhu et al., 2020). For the probes, the output is a probability distribution over quantiles, so we use accuracy of the most likely quantile.

Datasets and Subject Model For all our experiments our subject model is DEEPSICALER-1.5B (Luo et al., 2025) a small domain-specific reasoning model that is post-trained with reinforcement-learning from (Qwen et al., 2025).

We construct a reasoning dataset by randomly drawing questions from openr1-math (Face, 2025) and generating one reasoning trace per question by sampling from DEEPSICALER-1.5B, using parameters from (Luo et al., 2025) with temperature of 0.6 and top_p 0.95 resulting in a dataset of 25K traces.

We split our dataset into 3 parts: train, eval and out-of-distribution (ood) for reasoning traces above 16K tokens.

We assign a progress label to each token by quantizing $\frac{k}{m}$ into 10 equally-spaced quantiles $Q_i = [\frac{i-1}{10}, \frac{i}{10})$ for $i = 1, \dots, 10$. Each token at position k is labeled with quantile i where $\frac{k}{m} \in Q_i$.

3 EXPERIMENTS

To assess the extent to which existing reasoning model representations contain information about progress, we train two types of linear probes.

Let $h_{i,j}$ be the hidden state at the j 'th token of the i 'th reasoning trace.

1. A probe $f_{\text{probe}} : \mathbb{R}^d \rightarrow \mathbb{R}^Q$ that maps the hidden state $h_{i,j}$ at each token position to a Q -dimensional output representing the logits for each quantized progress bucket.
2. A concatenated probe $f_{\text{probe}}^{\text{concat}} : \mathbb{R}^{n*d+d} \rightarrow \mathbb{R}^Q$, where n is the number of question tokens used. $e_i \in \mathbb{R}^{n*d}$. The input is $[h_{i,j}; e_i]$, the concatenation of the current token's hidden state and the the question embedding derived from the final n tokens preceding the assistant token. Throughout our experiments, we set $n = 2$.

The model parameters θ are optimized using the cross-entropy loss over the Q progress buckets. For a given input $\mathbf{h}_{i,j}$ (either $h_{i,j}$ or $[h_{i,j}; e_i]$) and one-hot target label $y \in \{0, 1\}^Q$, the loss becomes:

$$\mathcal{L}(\theta) = -\frac{1}{\sum_{i=0}^B T_i} \sum_{i=0}^B \sum_{j=0}^{T_i} \sum_{q=1}^Q y_{i,j,q} \log \left(\frac{\exp(f_{\text{Probe}}(\mathbf{h}_{i,j})_q)}{\sum_{k=1}^Q \exp(f_{\text{Probe}}(\mathbf{h}_{i,j})_k)} \right)$$

where $f_{\text{Probe}}(\mathbf{h}_{i,j})_q$ denotes the logit for bucket q at position j in sequence i and $y_{i,j,q}$ is the indicator for the true bucket, B denotes the batch size and T_i the length of the reasoning trace in sequence i

We train single layer linear probes on layer 8, 16, 24 and obtain the hidden states by having DEEPSICALER-1.5B predict the next token. We train and evaluate on sequences below 16K tokens and withhold an out-of-distribution (OOD) dataset with sequences above 16K tokens. We use 10 quantiles.

The results in Table 1 show that probe performance is relatively consistent across layers, with accuracy around 30% and MAE around 1.75, moreover as illustrated from the heatmap 2 the probability mass is heavily concentrated around the ground truth label.

This suggests that the model's hidden states do encode some notion of progress, and that simple probes can extract this information to a large degree. However, performance drops substantially on

Table 1: Probe performance on DEEPSCLER-1.5B at different layers

Layer	Argmax Accuracy	MAE
8	0.30	1.75
16	0.29	1.76
24	0.31	1.74

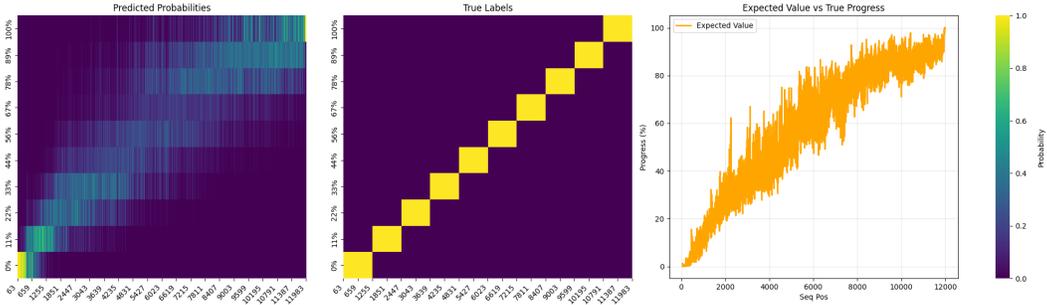


Figure 2: Visualization of linear probe prediction (Layer 16, $f_{\text{probe}}^{\text{concat}}$) for a 12K token sequence. Left: probability distribution over progress quantiles at different token positions. Center: True progress labels at different token positions. Right: The expected value over progress buckets, reducing prediction to a single number. The probe successfully tracks the upward trajectory of true progress throughout the reasoning sequence.

(OOD) sequences (Table 1), highlighting the challenge of generalization. The addition of question embeddings does not consistently improve results, indicating that most relevant information is already present in the hidden states at the given token position.

4 INCORPORATING PROGRESS ESTIMATION INTO THE REASONING TRACE

We start by performing supervised fine-tuning on our subject model, DEEPSCLER-1.5B, by introducing custom progress-tokens (`<progressbar>` and `</progressbar>`) and constructing a synthetic dataset by segmenting reasoning traces from the openr1-math dataset (Face, 2025) into discrete chunks and inserting special tokens every 250 tokens. Note that these reasoning traces are from a larger model (R1) and different from the dataset used for training the probes.

For each reasoning trace r containing thinking tokens between `<think>` and `</think>` markers, we:

1. Split the thinking portion into segments s_1, \dots, s_n using paragraph segmentation with Segment-Any-Text (Frohmann et al., 2024).
2. For each segment s_i , compute its token length l_i .
3. Calculate cumulative progress $p_i = \frac{\sum_{j=1}^i l_j}{\sum_{j=1}^n l_j}$.
4. Insert progress markers `<progressbar>` p_i `</progressbar>` after each segment every τ tokens. Here, p_i represents the numerical progress percentage (e.g., "23%") which is tokenized using the model's existing vocabulary rather than being treated as a special token. We explore alternative approaches using dedicated progress tokens in C.1.

We perform supervised fine-tuning using LoRA adapters (Hu et al., 2021) with peft (Mangrulkar et al., 2022). Where loss on the new special tokens are scaled by a coefficient α such that for a given sequence $x = (x_1, x_2, \dots, x_T)$ of length T , the loss becomes:

$$L = \frac{1}{S} \sum_{i=0}^S [(1 + (\alpha - 1) \cdot \mathbb{1}_{i \in \langle \text{progressbar} \rangle_{pos} \dots \langle \text{progressbar} \rangle_{pos}}) \cdot (-\log P_{\theta}(x_i | x_{<i}))]$$

We train with a context length of 2048 and $\alpha = 5$. We evaluate the fine-tuned model on Math500 (Face), AMC23 (zwhe99) and a subset of OlympiadBench (Hothan). A full sampled output can be found in (E).

Learning to backtrack We observe a monotonicity bias, where the model’s predictions are monotonically increasing. This causes errors to compound when the model makes an early optimistic prediction. We believe this is an artifact of the dataset construction where labels are monotonically increasing. Moreover we find that in long reasoning traces the model frequently starts to produce predictions above 100 percent once it has reached 100 percent 9.

To incentivize self-correction, we modify the ground truth progress labels \hat{p}_i by adding noise sampled from a normal distribution:

$$\tilde{p}_i = \hat{p}_i + \epsilon_i, \quad \text{where } \epsilon_i \sim \mathcal{N}(0, (0.1)^2) \quad (1)$$

This creates examples of non-monotonic behavior in progress, and although this provides a less clean signal, we find that this seems to improve performance.

Applying Reinforcement Learning To address the monotonicity bias and reduce invalid predictions above 100% in long context reasoning, we employ reinforcement learning to optimize the model’s progress prediction behavior.

We design three reward functions, evaluated on a batch of N model outputs $\mathcal{O} = \{o_1, \dots, o_N\}$ for a given question. For each output o_i , let $P_i = \{p_{i,1}, \dots, p_{i,T_i}\}$ denote the sequence of predicted progress percentages at each marker (with T_i markers), and $A_i = \{a_{i,1}, \dots, a_{i,T_i}\}$ the corresponding ground-truth progress percentages at those positions. Let y_i be the model’s boxed answer and a_i^* the ground-truth answer for output o_i .

- **Prediction Accuracy:** Mean Absolute Error (MAE) between the predicted and true progress percentages, clipped to $[0, 1]$:

$$R_{\text{pred}}(o_i) = 1 - \frac{1}{|P_i|} \sum_{j=1}^{|P_i|} |\text{clip}(p_{i,j}, 0, 1) - a_{i,j}| \quad (2)$$

where $\text{clip}(x, 0, 1) = \max(0, \min(x, 1))$.

- **Formatting Consistency:** Rewards regular insertion of progress markers approximately every τ tokens, and penalizes progress markers that are placed outside the `<think> ... </think>` region:

$$R_{\text{format}}(o_i) = 1 - \frac{1}{T_i - 1} \sum_{j=1}^{T_i-1} \min\left(\frac{|t_{i,j+1} - t_{i,j} - \tau|}{\tau}, 1\right) + P_{\text{outside}}(o_i) \quad (3)$$

where $t_{i,j}$ is the token index of the j -th progress marker in output o_i , T_i is the total number of progress markers in o_i , and $\tau = 250$ is the target interval. Note that this reward is maximized (equal to 1, before any outside-region penalties) when each consecutive pair of progress markers is exactly τ tokens apart, i.e., $t_{i,j+1} - t_{i,j} = \tau$ for all j . The penalty term $P_{\text{outside}}(o_i)$ is defined as $P_{\text{outside}}(o_i) = -\min(0.1 \cdot N_{\text{outside}}, 0.5)$, where N_{outside} is the number of progress markers that appear outside the `<think> ... </think>` region.

- **Mathematical Correctness:** Binary reward for correctly solving the underlying problem:

$$R_{\text{math}}(o_i) = \mathbb{1}[y_i = a_i^*] \quad (4)$$

where $\mathbb{1}[\cdot]$ is the indicator function, y_i is the model’s predicted answer, and a_i^* is the ground-truth answer.

The total reward for output o_i is a linear combination of individual rewards:

$$R(o_i) = \lambda_1 R_{\text{pred}}(o_i) + \lambda_2 R_{\text{format}}(o_i) + \lambda_3 R_{\text{math}}(o_i)$$

where $\lambda_1, \lambda_2, \lambda_3$ are weighting coefficients.

Table 2: Mathematical performance (accuracy %) on MATH500, AMC23, and OlympiadBench datasets

Model	MATH500	AMC23	OlympiadBench (Subset)
DEEPSCALER-1.5B	63.0	80.0	22.0
SFT (Naive Noisy)	64.2	60.0	19.3
SFT (No Noise)	63.0	62.5	13.3
RL (Naive Noise)	62.6	67.5	16.0
RL (No Noise)	61.4	72.5	16.0

Table 3: Progress prediction performance (MAE) across different model configurations and sequence length categories

Model	<16K Correct		<16K Incorrect		>16K Correct		>16K Incorrect	
	Raw	Clip	Raw	Clip	Raw	Clip	Raw	Clip
Random Baseline	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333
SFT (Noise)	0.104	0.104	0.136	0.132	0.206	0.206	0.288	0.256
SFT	0.224	0.139	0.265	0.157	0.796	0.491	4.027	0.386
SFT + RL	0.102	0.102	0.133	0.132	0.232	0.232	0.343	0.264
SFT (Noise) + RL	0.101	0.101	0.128	0.128	–	–	0.271	0.265

Objective As our policy gradient algorithm, we use GRPO (Shao et al., 2024) with modifications inspired by DAPO (Yu et al., 2025) and (Jaghour et al., 2025). The specific hyperparameters can be found in Appendix 6.

We find that performance on reasoning tasks drops after incorporating progress prediction capabilities into the base model, as evidenced by the results in Tables 3. This might be due to the model optimizing reward through prediction accuracy and formatting consistency at the expense of mathematical correctness. For example, a natural way to improve performance might be to produce shorter reasoning traces as these might be easier to predict 3, however we find that the mean length of the reasoning trace increases, not decreases.

Given our hardware constraints, we are forced to limit the completion length during RL to 8,096 tokens compared to the original model’s 24K which could impair reasoning performance.

More work is needed, to accurately measure to what extent simultaneous progress prediction hampers performance on the core task of reasoning. For example, during RL, the model could be incentivized to produce shorter inherently more predictable generations.

We also find that although RL overall does not seem to improve the mean progress prediction, it enables better progress prediction for longer sequences, by ensuring against the model guessing above 100 pct 10 9. In the ablations section of the appendixC, we suggest other approaches to lower prediction error; we find that using custom prediction tokens significantly improves performance and enables emergent backtracking behavior despite monotonic training data C.1.

4.1 COMPARATIVE EVALUATION: PROBE VS. FINE-TUNED LLM APPROACHES

Evaluation Protocol. Using the test used to train the probes, consisting of reasoning traces from (Luo et al., 2025). We segment random prefixes from our test set and prompt the model by appending a <progress_bar> tag at the selected position. We then perform autoregressive sampling until the model generates the prediction and the corresponding closing tag: </progress_bar>. This evaluation setup differs from the training regime, where the model has access to its previous progress predictions through the attention mechanism. To evaluate the probes we obtain the hidden states at a given position by having DEEPSCALER-1.5B predict the next token given the prefix.

Methodological Considerations. We evaluate both approaches on identical token positions within the same test sequences. Our probe-based methods output discrete probability distributions over 10 quantized progress classes: {0%, 11.1%, 22.2%, ..., 100%}. We convert this to discrete values

324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377

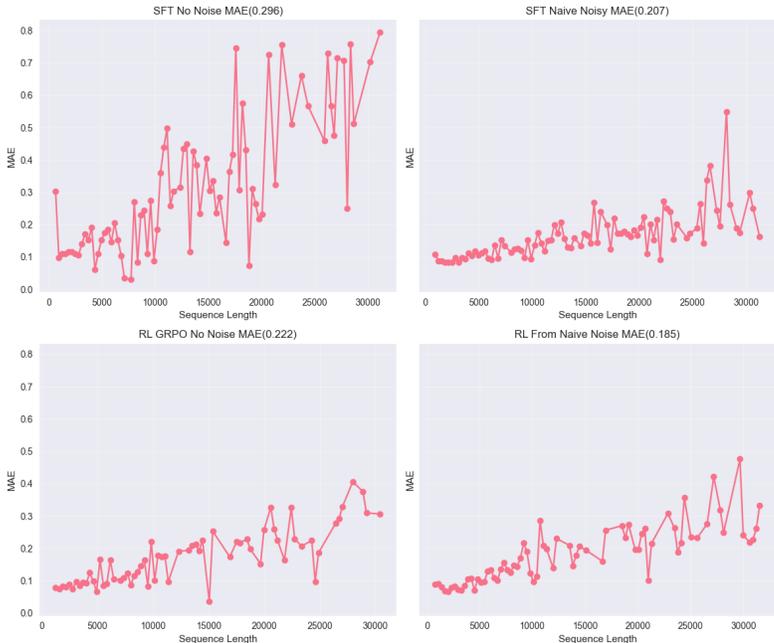


Figure 3: Prediction error as a function of length of the reasoning trace from which that prediction originated. We see that prediction error increases with sequence length. Predictions are clipped to be in interval (0,1)

Table 4: Comparison of MLP-probe (f_{probe}^{concat}) performance at different layers and LLM

Configuration	L8 (f_{probe})	L8 (f_{probe}^{concat})	L16 (f_{probe})	L16 (f_{probe}^{concat})	L24 (f_{probe})	L24 (f_{probe}^{concat})	SFT (No Noise)
MAE	0.1795	0.1443	0.2306	0.1737	0.3233	0.1970	0.2057

in $[0, 100]$ by computing the expected value $\mathbb{E}[p] = \sum_{i=0}^9 p_i \cdot (11.1i)$ where p_i is the predicted probability for class i , and rounding to closest integer. This ensures the output domains of both methods are identical.

Methodological Considerations. We evaluate both approaches on identical token positions within the same test sequences. Our probe-based methods output discrete probability distributions over 10 quantized progress classes: $\{0\%, 11.1\%, 22.2\%, \dots, 100\%\}$. We convert this to discrete values in $[0, 100]$ by computing the expected value $\mathbb{E}[p] = \sum_{i=0}^9 p_i \cdot (11.1i)$ where p_i is the predicted probability for class i , and rounding to closest integer. This ensures the output domains of both methods are identical.

5 HOW PREDICTABLE IS PROGRESS?

An interesting question is what is a reasonable lower bound on prediction error for progress estimation?

Given a question q , we take $i = 1 \dots n$ random prefixes C_k^i (the first k tokens) from the trace C and generate t rollouts for each, so that for the i 'th prefix we have $\tau_0^i \dots \tau_t^i$ rollouts, sampled from $P(\cdot | q, C_k^i)$. We sample with temperature 0.6 as recommended in (Luo et al., 2025), we argue that this is a fair assumption, as this is the temperature people are recommended to be using in production.

We set $t = 8$ and $n = 12$ and compare our results with $\frac{|C_k^i|}{|C|}$.

We examine 500 sequences in total, generated using questions from Math500. The dispersion of possible trajectories (lower left plot) shows the coefficient of variation over samples. As we condition

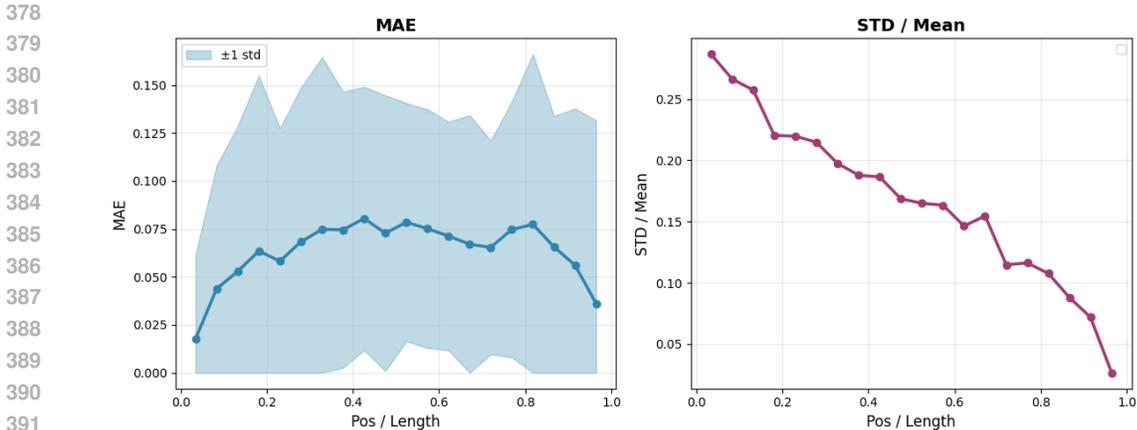


Figure 4: Relative position refers to (pos / sequence length). We observe that the mae is around 5%-10% suggesting this is a lower bound, for how well progress can be predicted. We also find that the coefficient of variation decreases monotonically with position, and thus the dispersion in possible CoT lengths decreases the further we are in the reasoning process. That is, as our observed prefix increases in length, it progressively constrains the distribution of possible trajectory completions.

on a larger number of tokens k , this value decreases. For $\hat{y}, y \in [0, 1]$ let \hat{y} be the estimated progress and y the true progress $\frac{k}{|C|}$ where $|C|$ is the total length of the complete trace. Let $P_{i,j} = \frac{|C_k^i|}{|C_k^i| + |T_j^i|}$

$$mae_i = \frac{1}{t} \sum_{j=1}^t \left| \frac{|C_k^i|}{|C|} - P_{i,j} \right| \text{ and } cv_i = \frac{\text{std}(P_{i,0}, \dots, P_{i,t})}{\text{mean}(P_{i,0}, \dots, P_{i,t})}$$

6 CONCLUSION AND FUTURE RESEARCH

In this paper, we explore methods for predicting progress in reasoning models. Our experiments demonstrate that current models encode a notion of progress in their internal representations, with an MLP probe achieving close to 30% accuracy over 10 classes, moreover the errors appear highly localized (MAE 1.75).

Motivated by these observations, we introduce a simple strategy for directly incorporating prediction into the llm through a 2-step process of SFT and RL - and demonstrate that at smaller scales (below 16K) the approach works significantly better than random guessing, even when limited to LoRA fine-tuning. However, the conflict between progress-prediction and reasoning quality remains unclear.

Future Research Directions We find, after having applied our post-training recipe, that model reasoning performance seems to degrade slightly. Future research should further study this phenomenon: is there an inherent tradeoff between these two tasks, or is degraded performance an artifact of our training methodology or limited computational resources?

This paper focuses on mathematical reasoning, which at face-value seems to be one of the most challenging domains for progress prediction (i.e., it seems harder to forecast how far we are from solving the Riemann hypothesis than how long it will take a person to go to the supermarket) — applying these principles to other domains such as research or multi-turn agents (Jin et al., 2025) could be fruitful area of research.

Another potential area of improvement is making the progress-update ”surprisal-based”, that is, the model learns to emit predictions at major/crucial points in the reasoning, when the model deviates from its prior rate of progress.

7 PRIOR WORK

7.1 SEARCH PROGRESS

Beside the foundational work of Thayer et al. (2012), Sudry & Karpas (2021) are the first to apply deep learning methods to progress prediction. They convert the problem into a supervised learning task, training an LSTM to predict progress, conditioned on prior search history.

Related work has been done in the field of neural theorem proving, where (Huang et al., 2025) train a language model to predict the number of steps remaining to reach a 'no goals state' (proof completion) in LEAN (Moura & Ullrich, 2021). (Ma et al., 2019; Zhu et al., 2020) apply progress estimation to vision-language navigation in robotics.

7.2 REINFORCEMENT LEARNING FOR LLMS

Reinforcement Learning is a learning approach in which an agent, acting in an environment, attempts to maximize future discounted rewards over a policy (Kaelbling et al., 1996; Sutton & Barto, 2018).

Reinforcement learning for LLMs was popularized with RLHF (Ouyang et al., 2022), in which a language model is optimized using feedback from a reward model trained on human preference data (Lambert et al., 2024). More recently, RL has been applied to train reasoning models (DeepSeek-AI et al., 2025).

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) has become popular for this purpose as it as opposed to Rafailov et al. (2024) removes the need for a value model while maintaining stable training dynamics.

Let q be a prompt and $\{o_0 \dots o_i \dots o_G\}$ rollouts sampled from $P_\theta(\cdot|q, o_{i,<t})$ where θ is our policy network. Then $\hat{A}_{i,t}$ is the advantage for token t in rollout i . We define $r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q, o_{i,<t})}$ as the probability ratio between the new and old policies.

GRPO Objective.

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = & \mathbb{E}_{q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(O|q)} \\ & \frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip} \left(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right) \right. \\ & \left. - \beta D_{\text{KL}}[\pi_\theta(\cdot|q) \parallel \pi_{\text{ref}}(\cdot|q)] \right\} \end{aligned} \quad (5)$$

The clip function restricts the probability ratio to the interval $[1 - \epsilon, 1 + \epsilon]$, with ϵ a small positive constant (e.g., 0.2), to avoid excessively large policy updates. KL divergence between the probability distributions is added as a regularization term.

Shao et al. (2024) introduced group relative advantages, where the advantage is computed as the group relative reward $\hat{A}_{i,t} = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$.

REFERENCES

- Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025. URL <https://arxiv.org/abs/2503.04697>.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang

- 486 Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong,
487 Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao,
488 Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang,
489 Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang,
490 Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L.
491 Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang,
492 Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng
493 Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanxia Zhao, Wen Liu, Wenfeng
494 Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan
495 Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang,
496 Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen,
497 Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li,
498 Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang,
499 Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan,
500 Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yudian Wang, Yue Gong, Yuheng Zou, Yujia
501 He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong
502 Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha,
503 Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang,
504 Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li,
505 Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen
506 Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.
URL <https://arxiv.org/abs/2501.12948>.
- 507 Hugging Face. Math500. URL [https://huggingface.co/datasets/HuggingFaceH4/](https://huggingface.co/datasets/HuggingFaceH4/MATH-500)
508 MATH-500.
- 509 Hugging Face. Openr1-math-220k. [https://huggingface.co/datasets/open-r1/](https://huggingface.co/datasets/open-r1/OpenR1-Math-220k)
510 OpenR1-Math-220k, 2025.
- 511 Markus Frohmann, Igor Sterner, Ivan Vulić, Benjamin Minixhofer, and Markus Schedl. Segment
512 any text: A universal approach for robust, efficient and adaptable sentence segmentation. In Yaser
513 Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on*
514 *Empirical Methods in Natural Language Processing*, pp. 11908–11941, Miami, Florida, USA,
515 November 2024. Association for Computational Linguistics. URL [https://aclanthology.](https://aclanthology.org/2024.emnlp-main.665)
516 [org/2024.emnlp-main.665](https://aclanthology.org/2024.emnlp-main.665).
- 517 Hothan. Olympiadbench. URL [https://huggingface.co/datasets/Hothan/](https://huggingface.co/datasets/Hothan/OlympiadBench)
518 OlympiadBench.
- 519 Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang,
520 and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL [https://](https://arxiv.org/abs/2106.09685)
521 arxiv.org/abs/2106.09685.
- 522 Suozhi Huang, Peiyang Song, Robert Joseph George, and Anima Anandkumar. Leanprogress:
523 Guiding search for neural theorem proving via proof progress prediction, 2025. URL [https://](https://arxiv.org/abs/2502.17925)
524 arxiv.org/abs/2502.17925.
- 525 Prime Intellect Team: Sami Jaghour, Justus Matterm, Jack Min Ong, Jannik Straube,
526 Manveer Basra, Aaron Pazdera, Matthew Di Ferrante, Kushal Thaman, Felix Gabriel,
527 Fares Obeid, Kemal Erdem, Michael KEBlinger, and Johannes Hagemann. Intellect-
528 2: A reasoning model trained through globally decentralized reinforcement learning,
529 2025. URL [https://storage.googleapis.com/public-technical-paper/](https://storage.googleapis.com/public-technical-paper/INTELLECT_2_Technical_Report.pdf)
530 [INTELLECT_2_Technical_Report.pdf](https://storage.googleapis.com/public-technical-paper/INTELLECT_2_Technical_Report.pdf). Preprint.
- 531 Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Serkan Arik, Dong Wang, Hamed Zamani, and
532 Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement
533 learning, 2025. URL <https://arxiv.org/abs/2503.09516>.
- 534 L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey, 1996. URL
535 <https://arxiv.org/abs/cs/9605103>.

- 540 Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan
541 Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian
542 Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein,
543 Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan.
544 Measuring ai ability to complete long tasks, 2025. URL <https://arxiv.org/abs/2503.14499>.
545
- 546 Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu,
547 Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi.
548 Rewardbench: Evaluating reward models for language modeling, 2024. URL <https://arxiv.org/abs/2403.13787>.
549
- 550 Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin
551 Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-
552 preview with a 1.5b model by scaling rl. <https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005b>
553
554 2025. Notion Blog.
555
- 556 Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming
557 Xiong. Self-monitoring navigation agent via auxiliary progress estimation, 2019. URL <https://arxiv.org/abs/1901.03035>.
558
- 559 Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin
560 Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. <https://github.com/huggingface/peft>, 2022.
561
562
- 563 Moura and Ullrich. The lean 4 theorem prover and programming language. In Platzer and Sutcliffe
564 (eds.), *Automated Deduction – CADE 28*, pp. 625–635, Cham, 2021. Springer International
565 Publishing. ISBN 978-3-030-79876-5.
- 566 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke
567 Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. sl: Simple test-time
568 scaling, 2025. URL <https://arxiv.org/abs/2501.19393>.
569
- 570 OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden
571 Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko,
572 Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally
573 Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich,
574 Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani,
575 Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao,
576 Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi,
577 Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong
578 Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts,
579 Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David
580 Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong,
581 Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang,
582 Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred
583 von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace
584 Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin,
585 Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian
586 O’Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever,
587 Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng,
588 Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñero Candela, Joe Palermo, Joel Parish,
589 Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan
590 Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl
591 Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin
592 Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus,
593 Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk,
Lukas Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko
Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz,
Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe,

- 594 Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang,
595 Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowd-
596 hury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg
597 Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias,
598 Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny
599 Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi
600 Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago
601 Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani
602 Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir
603 Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted
604 Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng,
605 Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie
606 Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou,
607 Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai,
608 Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai ol system card,
609 2024. URL <https://arxiv.org/abs/2412.16720>.
- 610 Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong
611 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton,
612 Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and
613 Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL
614 <https://arxiv.org/abs/2203.02155>.
- 615 Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan
616 Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang,
617 Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin
618 Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi
619 Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan,
620 Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL
621 <https://arxiv.org/abs/2412.15115>.
- 622 Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea
623 Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL
624 <https://arxiv.org/abs/2305.18290>.
- 625 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
626 Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of
627 mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- 628
629
- 630 Matan Sudry and Erez Karpas. Learning to estimate search progress using sequence of states. In
631 *ICAPS 2021 Workshop on Heuristics and Search for Domain-independent Planning*, 2021. URL
632 <https://openreview.net/forum?id=W9O6ueH5U37>.
- 633
634 Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford
635 Book, Cambridge, MA, USA, 2018. ISBN 0262039249.
- 636 Jordan T. Thayer, Roni Stern, and Levi H. Lelis. Are we there yet? – estimating search progress. In
637 *Proceedings of the International Symposium on Combinatorial Search (SOCS)*. AAAI Press, 2012.
638
- 639 Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong
640 Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi
641 Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiase Chen, Jiangjie Chen, Chengyi
642 Wang, Hongli Yu, Weinan Dai, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying
643 Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source
644 llm reinforcement learning system at scale, 2025. URL <https://arxiv.org/abs/2503.14476>.
- 645
646 Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning
647 models know when they’re right: Probing hidden states for self-verification, 2025. URL <https://arxiv.org/abs/2504.05419>.

Table 5: Comparison of conditional vs marginal prediction performance on AMC23

Condition	MAE (all predictions)	MAE (filtered)
Unconditioned	0.218	0.176
Conditioned	0.297	0.173

Table 6: Hyperparameters used in all experiments

Parameter	Value
β	0
ϵ_{high}	0.28
ϵ	0.2
learning rate	1×10^{-6}
lr scheduler	constant with warmup
warmup steps	25
num train epochs	1
num generations	4
temperature	1.0
max steps	500
bf16	True
max grad norm	0.1
max completion length	8096
δ	4

Fengda Zhu, Yi Zhu, Xiaojun Chang, and Xiaodan Liang. Vision-language navigation with self-supervised auxiliary reasoning tasks, 2020. URL <https://arxiv.org/abs/1911.07883>.
 zwhe99. Amc 2023. URL <https://huggingface.co/datasets/zwhe99/amc23>.

A DO PRIOR PREDICTIONS MATTER?

One interesting question is: what is the difference between modeling conditional (where we can attend to prior predictions) vs the marginal (where we don't have access to prior ones)?

We study this for our supervised fine-tuned model (sft-no-noise) on the reasoning traces generated on Amc23.

As vLLM does not support custom masking, we use str-replace to remove any prior predictions, then prompt each model with the prefix leading up to the current position with a `<progressbar>` marker.

The independent estimate achieves better overall performance (0.218 vs 0.297 MAE), but when filtering for invalid predictions above 100%, performance is comparable (0.176 vs 0.173). Intuitively, the conditional entropy cannot increase when we condition on more variables, and so a robust predictor, we would expect to be at least as good if not better. We also find that for domains in which the model does not think about predicting above 100%, conditioning on past predictions leads to slightly better performance.

B HYPERPARAMETERS

C ABLATIONS

C.1 USING CUSTOM PREDICTION TOKENS ENABLES BACKTRACKING

Instead of having the tokens within (`<progressbar></progressbar>`) be normal tokens, for example leading to a prediction like '`<progressbar>10%</progressbar>`' where '1', '0'

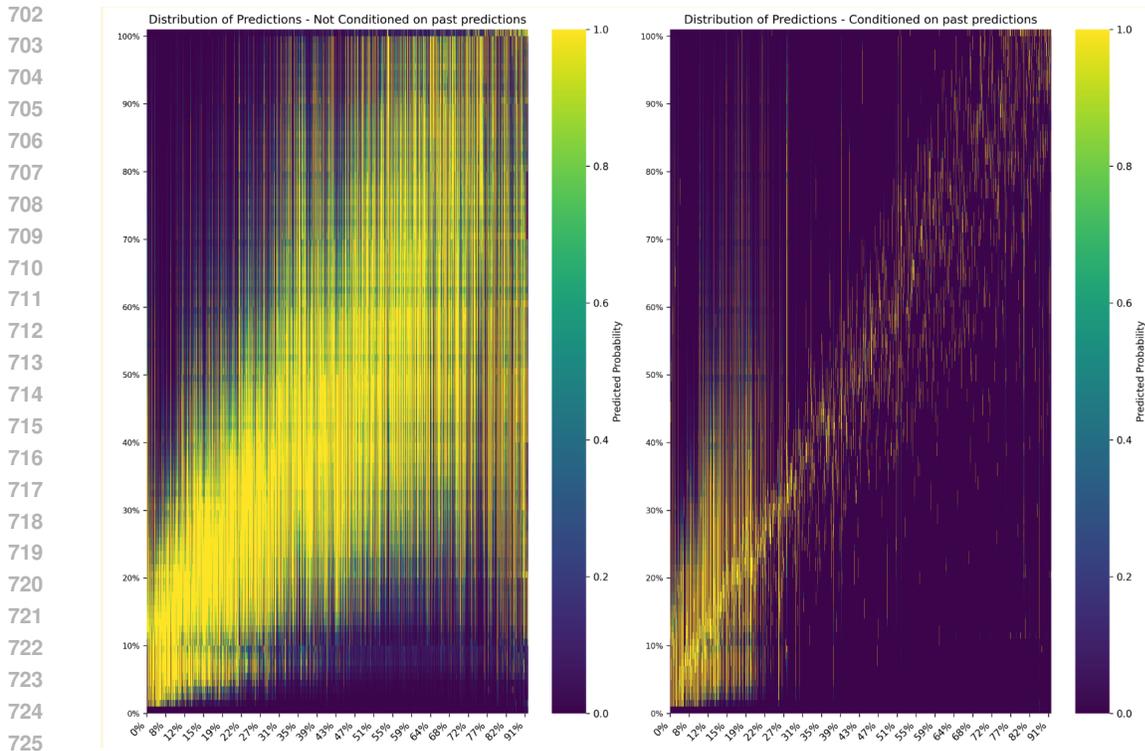


Figure 5: Heatmap visualization of predicted probability distributions across multiple sequences. Each column represents data points at a specific actual progress position (x-axis), while the y-axis shows the predicted progress bins (0%-100%). The color intensity indicates the probability mass assigned to each prediction bin, aggregated across all sequences in the dataset

and `'%` are separate tokens, we construct tokens `[0%] ... [100%]`. Thus the model would instead be trained to predict `<progressbar>[10%]</progressbar>`. This has the additional benefit that we can obtain the probability distribution over our labels 0% - 100% in one step.

Our hypothesis is that reusing tokens like `'1' '%'` for custom prediction tasks as well as for the standard language modeling, confuses the model, thereby leading to problems like the model predicting above 100%.

Model	Dataset	MAE	MAE (clipped)	Accuracy
with special tokens	math500	0.2012	-	0.6460
sft (No Noise)	math500	0.2905	0.2088	0.6300
with special tokens	ommath	0.2374	-	0.1667
sft (No Noise)	ommath	0.3395	0.2228	0.1533
with special tokens	amc23	0.2121	-	0.7000
sft (No Noise)	amc23	0.2990	0.2561	0.6250

Table 7: Performance comparison between SFT model with and without custom prediction tokens

Interestingly, we find that despite being trained on a dataset that only shows monotonically increasing progress markers, the model is able to backtrack on its prior predictions. We find this to be the case despite not being explicitly trained to backtrack, where the model is able to exhibit backtracking behavior.

Table 8: Performance comparison between models with custom predictions tokens with and without masking schedule

Model	Dataset	MAE	Accuracy
w/o masking schedule	math500	0.2012	0.6460
w/ masking schedule	math500	0.1874	0.6240
w/o masking schedule	ommath	0.2374	0.1667
w/ masking schedule	ommath	0.2141	0.1533
w/o masking schedule	amc23	0.2121	0.7000
w/ masking schedule	amc23	0.1958	0.7750

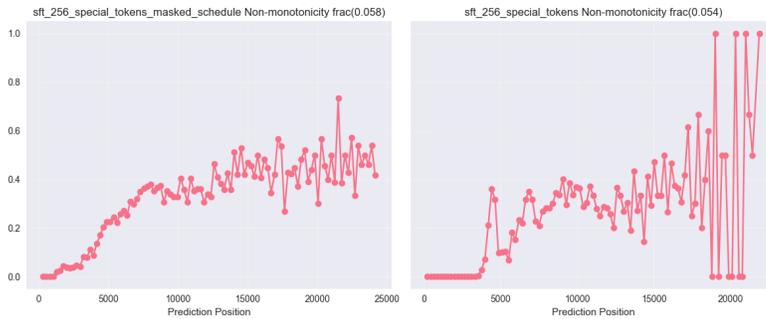


Figure 6: Ratio of non monotonic / monotonic predictions as a function of sequence length.

C.2 MASKING SCHEDULES

We apply a cosine warmup schedule from 0 to 0.5 where all tokens inside `<progressbar></progressbar>` are masked out with probability up to 50%. Our hypothesis is that this can act as a form of regularization, ensuring robustness.

We find that the model’s prediction error is lower while reasoning performance stays roughly constant.

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

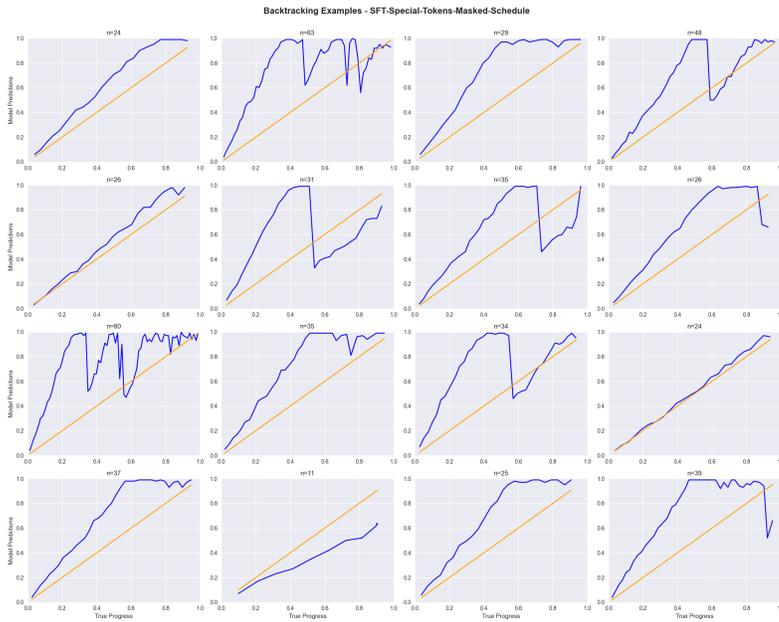


Figure 7: 16 examples of backtracking, the line with slope 1 indicates the perfect prediction for the model trained with special tokens and a masking schedule

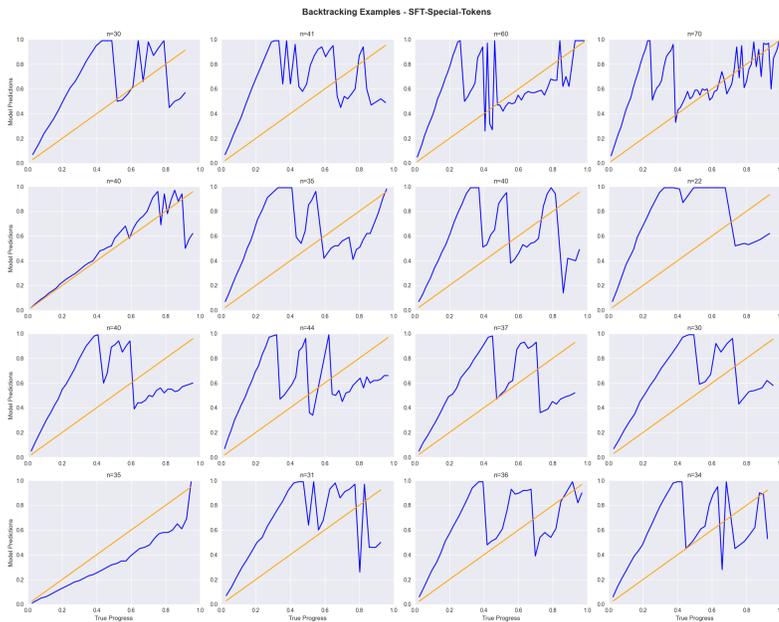


Figure 8: The figure shows examples of predictions during different reasoning trajectories

D GRAPHS

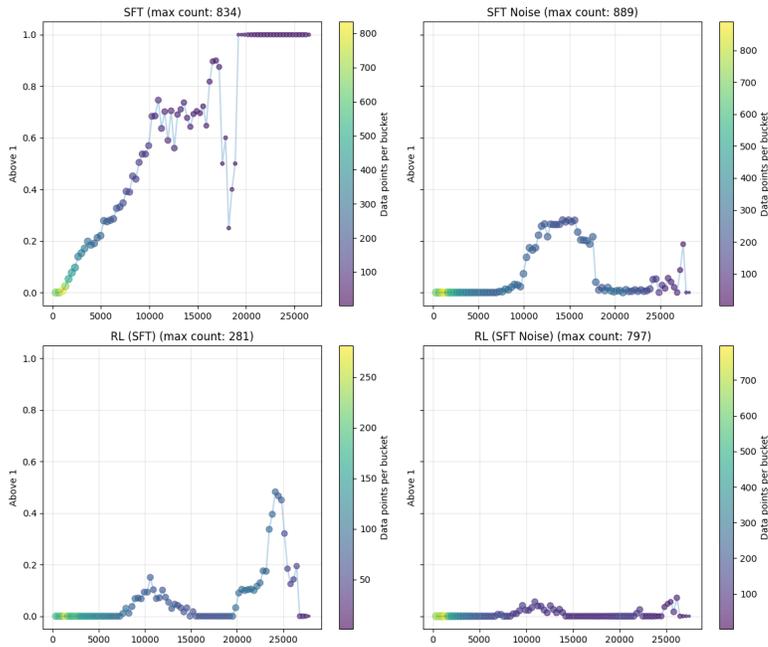


Figure 9: Fraction of invalid predictions above 100% on AMC23 dataset

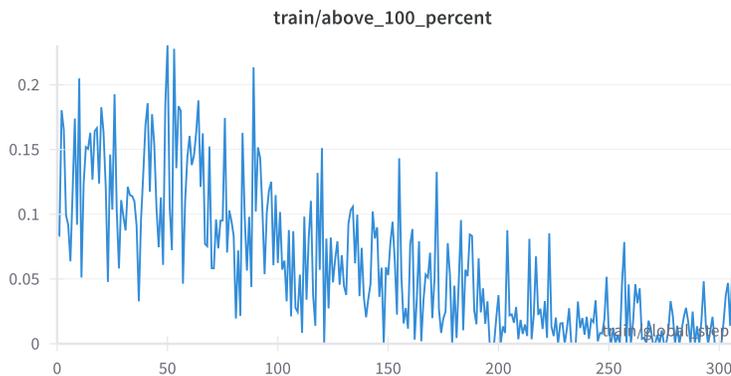


Figure 10: Percent of model predictions above 100 percent, during training

E EXAMPLE

Below is an example of a reasoning trace from the sft (No Noise) model.

```
<think>\n'
'Okay, so I need to figure out how many different values I can get by '
'inserting parentheses in the expression 2·3·4·5 + 1. The problem says that '
'changing the order of operations by adding parentheses can give different '
'results. For example, they gave (2·(3·4))·(5+1) = 144, which is different '
'from the original 121. The question is asking for the total number of '
'distinct values possible through all such insertions.\n'
```

918 'First, let me recall that parentheses can change the order in which '
919 'operations are performed. Since multiplication has higher precedence than '
920 'addition, inserting parentheses can force certain operations to happen '
921 'first. The original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. If I just follow the order '
922 '"without parentheses, it's $(2 \cdot 3 \cdot 4 \cdot 5) + 1 = 120 + 1 = 121$. But by inserting "
923 'parentheses, we can group different operations together, which might lead to '
924 'different results.\n'

925 'The key here is to consider all possible ways to parenthesize the '
926 'expression. Each insertion of parentheses can potentially change the order '
927 'of operations. <progressbar>2%</progressbar> The original expression has '
928 'four numbers: 2, 3, 4, 5, and the +1 at the end. So, how many different '
929 'parenthesizations are there?\n'

930 'I remember that for an expression with n numbers and k operations, the '
931 'number of different parenthesizations is given by the Catalan numbers. But '
932 'here, the operations are fixed: multiplication and addition. However, the '
933 'problem allows inserting parentheses to change the order of operations, but '
934 '"the numbers and operations are fixed. So maybe it's similar to inserting "
935 'parentheses in a binary tree structure.\n'

936 "Wait, let's think. The original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. The operations "
937 'are multiplication between 2 and 3, 3 and 4, 4 and 5, and then addition of '
938 '1. But inserting parentheses can change where the additions are performed.\n'
939 'Alternatively, maybe we can model this as a binary tree where each '
940 'multiplication is a node with two children, and the addition is a node with '
941 'one child. But since we have fixed operations, the structure is fixed. '
942 'However, inserting parentheses can group different operations together.\n'

943 'Wait, perhaps another approach: <progressbar>4%</progressbar> The original '
944 'expression is a sequence of multiplications followed by an addition. If we '
945 'insert parentheses, we can group some multiplications together, which might '
946 'affect the order in which additions happen.\n'

947 'Let me consider all possible ways to parenthesize the expression. The '
948 'original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. The possible parenthesizations would '
949 'involve grouping different parts together. For example:\n'

950 '1. Group all multiplications first: $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'
951 '2. Group some multiplications first and then the addition: $(2 \cdot 3 \cdot 4) \cdot (5 + 1) =$ '
952 ' $24 \cdot 6 = 144$ \n'

953 '3. Alternatively, maybe group the addition first? But addition has lower '
954 '"precedence, so unless we have parentheses around the addition, it's not "
955 'possible. So the addition must be done after all multiplications. Wait, but '
956 'the original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. If we insert parentheses to group '
957 '"the addition first, but since addition is lower precedence, we can't do "
958 'that. <progressbar>6%</progressbar> So the addition must be done after all '
959 'multiplications.\n'

960 'Wait, but the original expression is written as $2 \cdot 3 \cdot 4 \cdot 5 + 1$. If we insert '
961 'parentheses to group the addition first, it would have to be something like '
962 '" $2 \cdot 3 \cdot 4 \cdot (5 + 1)$, but that's already parenthesized as $((2 \cdot 3 \cdot 4) \cdot 6)$ which is "
963 'different. Wait, but the original expression is written as $2 \cdot 3 \cdot 4 \cdot 5 + 1$. So '
964 'maybe the structure is that all multiplications are done first, and then the '
965 '"addition. But that's the original expression.\n"

966 'But the problem allows inserting parentheses, so we can group different '
967 'operations. For example, if we group $5 + 1$ first, but addition has lower '
968 '"precedence. So unless we have parentheses around the addition, it's not "
969 'possible. Therefore, the addition must be done after all multiplications. '
970 'Therefore, the only way to group operations is to group some multiplications '
971 'together, which would change the order in which the addition is performed.\n'

972 "Wait, maybe not. Let's think of the expression as a sequence of operations. "
973 '<progressbar>8%</progressbar> The original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. The '
974 'operations are multiplication and addition. The structure is: $2 \times 3 \times 4 \times 5$ '
975 '+ 1. If we insert parentheses, we can group some multiplications first, '
976 'which would change the order in which additions happen. For example, if we '

972 'group 2·3 first, then multiply by 4, then multiply by 5, and then add 1. But '
 973 "that's the same as the original expression. Alternatively, if we group 3·4 "
 974 "first, then multiply by 5, then add 1. But again, that's the same as the "
 975 'original expression. Wait, perhaps inserting parentheses can allow us to '
 976 'perform operations in different orders.\n'
 977 'Wait, perhaps the key is that when you have multiple multiplication '
 978 'operations, you can choose which ones to group first. For example, in the '
 979 'original expression, there are three multiplication operations: 2×3, 3×4, '
 980 '4×5. If we group some of these first, the addition might be done after. Let '
 981 'me try to list all possible parenthesizations.\n'
 982 ' <progressbar>10%</progressbar> The original expression is 2·3·4·5 + 1. '
 983 "Let's consider the possible groupings:\n"
 984 '1. All multiplications first: ((2·3)·4)·5 + 1 = 120 + 1 = 121\n'
 985 '2. Group 2·3·4 first: (2·3·4)·5 + 1 = 24·5 + 1 = 120 + 1 = 121. Wait, that's "
 986 "the same as before. Hmm. So that's not different.\n"
 987 'Wait, maybe grouping 3·4 first: 2·(3·4)·5 + 1 = 2·24·5 + 1 = 48·5 + 1 = 240 '
 988 '+ 1 = 241. Wait, that's different. So that's another value.\n"
 989 'Similarly, grouping 4·5 first: 2·3·(4·5) + 1 = 2·3·20 + 1 = 6·20 + 1 = 120 + '
 990 '1 = 121. <progressbar>12%</progressbar> Again, same as original.\n'
 991 "Alternatively, grouping 2·3·4·5 first: but that's just 120, then add 1. But "
 992 "the original expression is written as 2·3·4·5 + 1, so perhaps we can't group "
 993 'the entire expression first. So maybe the possible groupings are:\n'
 994 '- Group 2·3 first: ((2·3)·4)·5 + 1 = 120 + 1 = 121\n'
 995 '- Group 3·4 first: 2·(3·4)·5 + 1 = 24·5 + 1 = 121\n'
 996 '- Group 4·5 first: 2·3·(4·5) + 1 = 6·20 + 1 = 121\n'
 997 '- Group 2·3·4 first: (2·3·4)·5 + 1 = 24·5 + 1 = 121\n'
 998 '- Group 2·3 first and 4·5 first: 2·(3·4·5) + 1 = 2·60 + 1 = 121\n'
 999 '- Group 3·4 first and 5: 2·3·(4·5) + 1 = same as above.\n'
 1000 ' <progressbar>15%</progressbar> Wait, maybe there are more possibilities. '
 1001 "Let's think step by step.\n"
 1002 "The expression is 2·3·4·5 + 1. Let's consider the possible ways to "
 1003 'parenthesize. The operations are multiplication and addition. The structure '
 1004 'is a binary tree with four multiplications and one addition. However, the '
 1005 "addition is at the end, so it's a left-associative operation. But by "
 1006 'inserting parentheses, we can change the order in which the additions are '
 1007 'performed.\n'
 1008 'Wait, perhaps we can model this as inserting parentheses in different '
 1009 'places. Let me recall that in such expressions, the number of different '
 1010 'parenthesizations is given by the Catalan numbers, but here, since the '
 1011 'operations are fixed, maybe the number is different.\n'
 1012 'Wait, the original expression is 2·3·4·5 + 1. The Catalan number for n=4 is '
 1013 "14, but that's for different parenthesizations with all possible operations. "
 1014 'However, in our case, the operations are fixed: multiplication and addition. '
 1015 'So maybe the number of distinct values is less.\n'
 1016 'Alternatively, perhaps the problem is similar to evaluating the expression '
 1017 "with different groupings. <progressbar>17%</progressbar> Let's try to list "
 1018 'all possible parenthesizations.\n'
 1019 "First, note that the original expression is 2·3·4·5 + 1. Let's consider all "
 1020 'possible ways to group the multiplications and additions.\n'
 1021 '1. All multiplications first: ((2·3)·4)·5 + 1 = 120 + 1 = 121\n'
 1022 '2. Group 2·3 first, then multiply by 4, then add 5: 2·3·(4·5) + 1 = 2·3·20 + '
 1023 '1 = 120 + 1 = 121\n'
 1024 '3. Group 3·4 first, then multiply by 5, then add 1: (2·3·4)·5 + 1 = 24·5 + 1 '
 1025 '= 121\n'
 1026 '4. Group 4·5 first, then multiply by 2, then add 1: 2·3·(4·5) + 1 = same as '
 1027 'above, 121\n'
 1028 'Wait, so all these groupings result in 121. <progressbar>19%</progressbar> '
 1029 "That can't be right. The problem states that inserting parentheses can give "

1026 'different results. So maybe there are other groupings.\n'

1027 'Wait, perhaps if we group some multiplications and then the addition. For '

1028 "example, $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 2 \cdot 24 \cdot 5 + 1 = 240 + 1 = 241$. That's a different "

1029 "value. So that's another value.\n"

1030 'Similarly, $3 \cdot (2 \cdot 4) \cdot 5 + 1 = 3 \cdot 8 \cdot 5 + 1 = 120 + 1 = 121$. Hmm, same as before.\n'

1031 'Alternatively, $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 2 \cdot 3 \cdot 20 + 1 = 120 + 1 = 121$. Again, same.\n'

1032 'Wait, so perhaps the key is that when you group different multiplications '

1033 "together, you can get different intermediate results. Let's see.\n"

1034 'Let me try to list all possible parenthesizations. The expression is $2 \cdot 3 \cdot 4 \cdot 5$ '

1035 '+ 1. The operations are multiplication and '

1036 'addition. The original expression is left-associative, so all '

1037 'multiplications happen before the addition. However, by inserting '

1038 'parentheses, we can change the order in which multiplications are done.\n'

1039 'Possible parenthesizations:\n'

1040 '1. $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'

1041 '2. $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 2 \cdot 24 \cdot 5 + 1 = 240 + 1 = 241$ \n'

1042 '3. $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 2 \cdot 3 \cdot 20 + 1 = 120 + 1 = 121$ \n'

1043 '4. $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 2 \cdot 3 \cdot 4 \cdot 6 = 144$ \n'

1044 '5. $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 2 \cdot 60 + 1 = 121$ \n'

1045 '6. $(2 \cdot 3) \cdot 4 \cdot (5 + 1) = 12 \cdot 4 \cdot 6 = 288 + 1$? Wait, '

1046 'no. Wait, if we group $5 + 1$ first, but addition has lower precedence. Wait, '

1047 'unless we insert parentheses around the addition. But in the original '

1048 'expression, the addition is at the end. So unless we have parentheses around '

1049 "the entire expression, but that's not allowed.\n"

1050 'Wait, maybe there are more possibilities. Let me think again.\n'

1051 'Alternatively, perhaps the expression can be parenthesized in different ways '

1052 'where different multiplications are grouped. For example:\n'

1053 '1. Group $2 \cdot 3$ first: $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'

1054 '2. Group $3 \cdot 4$ first: $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 2 \cdot 24 \cdot 5 + 1 = 240 + 1 = 241$ \n'

1055 '3. Group $4 \cdot 5$ first: $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 2 \cdot 3 \cdot 20 + 1 = 120 + 1 = 121$ \n'

1056 '4. Group $2 \cdot 3 \cdot 4$ first: $(2 \cdot 3 \cdot 4) \cdot 5 + 1 = 24 \cdot 5 + 1 = 120 + 1 = 121$ \n'

1057 '5. Group $2 \cdot 3 \cdot 4 \cdot 5$ first: $2 \cdot 3 \cdot 4 \cdot 5 + 1 = 120 + 1 = 121$ \n'

1058 '6. Group $2 \cdot 3$ first and $4 \cdot 5$ first: $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 2 \cdot 60 + 1 = 120 + 1 = 121$ \n'

1059 '7. Group $3 \cdot 4$ first and 5 : $2 \cdot 3 \cdot (4 \cdot 5) + 1 =$ same as above, 121 \n'

1060 '8. Group $2 \cdot 3 \cdot 4$ first and 5 : $2 \cdot (3 \cdot 4) \cdot 5 + 1 =$ same as above, 241 \n'

1061 " Wait, but that's only 7 distinct values: "

1062 '121, 144, 241. But that seems too few. The problem says "how many values can '

1063 'be obtained", and the example gives 144, which is one of them. Maybe there '

1064 'are more.\n'

1065 'Wait, perhaps I need to consider all possible ways of parenthesizing the '

1066 'multiplication and addition. The original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. Let me '

1067 'think of the possible ways to group the multiplications.\n'

1068 'The multiplication operations are between $2 \cdot 3$, $3 \cdot 4$, $4 \cdot 5$. The addition is '

1069 'between the result of $2 \cdot 3 \cdot 4 \cdot 5$ and 1. But inserting parentheses can change '

1070 'the order of operations. For example, if we group $2 \cdot 3$ first, then multiply '

1071 "by 4, then multiply by 5, then add 1. But that's the same as the original "

1072 'expression. Alternatively, if we group $3 \cdot 4$ first, then multiply by 5, then '

1073 'add 1. That would be $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 2 \cdot 24 \cdot 5 + 1 = 240 + 1 = 241$.\n'

1074 " Alternatively, if we group $4 \cdot 5$ first, then '

1075 "multiply by 2, then add 1. That's $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 2 \cdot 3 \cdot 20 + 1 = 120 + 1 =$ "

1076 " 121 . So that's the same as the original.\n"

1077 'Alternatively, if we group $2 \cdot 3 \cdot 4$ first, then multiply by 5, then add 1. '

1078 'That's $2 \cdot 3 \cdot 4 \cdot 5 + 1 = 120 + 1 = 121$.\n"

1079 'Alternatively, if we group $2 \cdot 3$ first, then multiply by 4, then add 5, but '

1080 "that's the same as the original.\n"

1081 'Wait, perhaps the key is that the addition is after all multiplications. But '

1082 'if we can group different multiplications together, we can change the order. '

1083 'Let me think of the expression as a sequence of operations. The original is '

1084 ' $2 * 3 * 4 * 5 + 1$. If we insert parentheses to group different '

1080 'multiplications, we can create different intermediate results.\n'

1081 ' <progressbar>31%</progressbar> Wait, perhaps another way: the expression '

1082 ' can be written as $2 * 3 * 4 * 5 + 1$. If we insert parentheses to group the '

1083 ' first two multiplications, then multiply by 4 and 5, then add 1. But that's "

1084 ' the same as the original. Alternatively, if we group the last two '

1085 ' multiplications first, then multiply by the first two, then add 1. That '

1086 ' would be $((2 * 3) * (4 * 5)) + 1 = 12 * 20 + 1 = 240 + 1 = 241$.\n'

1087 ' Alternatively, if we group the first multiplication and the last two '

1088 ' multiplications. Wait, but how?\n'

1089 ' Wait, perhaps the possible parenthesizations are:\n'

1090 ' 1. $((2*3)*4)*5 + 1 = 120 + 1 = 121$ \n'

1091 ' 2. $2*(3*4)*5 + 1 = 240 + 1 = 241$ \n'

1092 ' 3. $2*3*(4*5) + 1 = 120 + 1 = 121$ \n'

1093 ' 4. $2*3*4*(5 + 1) = 240 + 1 = 241$ \n'

1094 ' <progressbar>33%</progressbar> 5. $2*(3*4*5) + 1 = 2*60 + 1 = 120 + 1 = 121$ \n'

1095 ' 6. $(2*3)*4*5 + 1 = 12*4*5 + 1 = 240 + 1 = 241$ \n'

1096 "Wait, but these are all the same values: 121 and 241. That can't be right. "

1097 "The problem must have more possible values. Maybe I'm missing some.\n"

1098 "Wait, perhaps there's another way to group. For example, $2*3*4*5 + 1$ can be "

1099 "grouped as $2*(3*(4*5 + 1))$ or something. Let's try that.\n"

1100 ' If we group $4*5$ first, then add 1, then multiply by 3, then multiply by 2. '

1101 ' That would be $2*3*(4*5 + 1) = 2*3*(20 + 1) = 2*3*21 = 63 + 1 = 64$ '

1102 ' <progressbar>35%</progressbar> Wait, no. Wait, the expression is $2*3*4*5 +$ '

1103 ' 1. If we group $4*5$ first, then add 1, then multiply by 3 and 2. Wait, but '

1104 ' the original expression is written as $2*3*4*5 + 1$, so the addition is at the '

1105 ' end. If we insert parentheses around $4*5 + 1$, then multiply by 3 and 2. But '

1106 ' that would be $2*3*(4*5 + 1) = 2*3*21 = 126 + 1 = 127$? Wait, no. Wait, the '

1107 ' original expression is $2*3*4*5 + 1$. If we group $4*5$ first, then add 1, then '

1108 ' multiply by 3 and 2. But how does that work? Let's parse it.\n"

1109 ' If we write $2*3*4*5 + 1$ as $2*3*(4*5 + 1)$. Then compute $4*5 + 1$ first, which '

1110 ' is $20 + 1 = 21$. <progressbar>37%</progressbar> Then multiply by 3 and 2: '

1111 ' $2*3*21 = 6*21 = 126$. Then add 1? Wait, no. Wait, the original expression is '

1112 ' $2*3*4*5 + 1$. If we group $4*5 + 1$ first, then multiply by 3 and 2, but the '

1113 ' original expression is $2*3*4*5 + 1$. So inserting parentheses around $4*5 + 1$ '

1114 ' would require changing the original structure. Let me see:\n'

1115 ' Original: $2*3*4*5 + 1$ \n'

1116 ' If we want to group $4*5 + 1$ first, we need to write it as $2*3*(4*5 + 1)$. But '

1117 ' in the original expression, the addition is separate. So inserting '

1118 ' parentheses around $4*5 + 1$ would require modifying the original expression. '

1119 ' However, the original expression is written as $2*3*4*5 + 1$. So inserting '

1120 ' parentheses around $4*5 + 1$ would mean changing it to $2*3*(4*5 + 1)$, which '

1121 ' would be different. <progressbar>40%</progressbar> But in that case, the '

1122 ' value would be $2*3*21 = 6*21 = 126$, then add 1? Wait, no. Wait, the original '

1123 ' expression is $2*3*4*5 + 1$. If we insert parentheses around $4*5 + 1$, the '

1124 ' expression becomes $2*3*(4*5 + 1)$. Then, according to order of operations, '

1125 ' multiplication is done before addition, so first compute $4*5 + 1 = 21$, then '

1126 ' multiply by 3 and 2: $2*3*21 = 6*21 = 126$. But in the original expression, '

1127 ' the +1 is outside the parentheses. So if we group $4*5 + 1$ first, the value '

1128 ' would be 126, but the original expression is $2*3*4*5 + 1$. So the difference '

1129 ' is whether the +1 is after the entire multiplication or not. If we insert '

1130 ' parentheses around $4*5 + 1$, then the expression becomes $2*3*(4*5 + 1)$. '

1131 ' <progressbar>42%</progressbar> If we don't insert parentheses, it's $2*3*4*5$ "

1132 ' + 1. Therefore, inserting parentheses around $4*5 + 1$ changes the value. So '

1133 ' that's a new value. So that's 126. Then, are there other groupings?\n"

1134 "Wait, let's check another grouping. Suppose we group $2*3$ first, then "

1135 "multiply by 4 and 5, then add 1. But that's the original expression. "

1136 "Alternatively, group $2*3*4$ first, then multiply by 5, then add 1. That's "

1137 " $24*5 + 1 = 120 + 1 = 121$. So that's the same as original. Alternatively, "

1138 "group $2*3*4*5$ first, then add 1. That's $120 + 1 = 121$. So that's the same.\n"

1134 "Alternatively, group $3 \cdot 4$ first, then multiply by 5, then add 1. That's "

1135 " $2 \cdot 24 \cdot 5 + 1 = 240 + 1 = 241$. <progressbar>44%</progressbar> So that's "

1136 'another value.\n'

1137 'Wait, but if we group $4 \cdot 5$ first, then multiply by 2 and 3, then add 1. That '

1138 'would be $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 12 \cdot 20 + 1 = 240 + 1 = 241$. So same as before.\n'

1139 "Alternatively, group $2 \cdot 3 \cdot 4$ first, then multiply by 5, then add 1. That's "

1140 ' $24 \cdot 5 + 1 = 120 + 1 = 121$.\n'

1141 "Alternatively, group $2 \cdot 3$ first, then multiply by 4 and 5, then add 1. That's "

1142 'the original expression. So same result.\n'

1143 "Alternatively, group $2 \cdot 3 \cdot 4 \cdot 5$ first, then add 1. That's $120 + 1 = 121$.\n"

1144 'Wait, so so far, we have two values: 121 and 241. But the problem says "how '

1145 'many values can be obtained", and the example gives 144. '

1146 '<progressbar>46%</progressbar> So there must be more.\n'

1147 'Wait, perhaps I need to consider different groupings where different '

1148 'multiplications are combined in different orders. Let me try another '

1149 'approach.\n'

1149 "The original expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. Let's consider all possible ways to "

1150 'group the multiplications. Each multiplication can be grouped or not. '

1151 'However, the addition is fixed. So the possible groupings would involve '

1152 'inserting parentheses around different parts of the expression.\n'

1153 'Possible groupings:\n'

1154 '1. $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'

1155 '2. $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 240 + 1 = 241$ \n'

1156 '3. $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 120 + 1 = 121$ \n'

1157 '4. $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 240 + 1 = 241$ \n'

1158 '5. $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 2 \cdot 60 + 1 = 120 + 1 = 121$ \n'

1159 '6. <progressbar>49%</progressbar> Wait, but if we group $5 + 1$ first, then '

1160 'multiply by 2 and 3 and 4. But in the original expression, the addition is '

1161 'separate. So inserting parentheses around $5 + 1$ would change the order. For '

1162 'example, if we write $2 \cdot 3 \cdot 4 \cdot (5 + 1)$, that's different from the original. So "

1163 'that gives $2 \cdot 3 \cdot 4 \cdot 6 = 144$. So that's a new value.\n"

1164 'Similarly, if we group $4 \cdot 5$ first, then multiply by 2 and 3, then add 1. That '

1165 'would be $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 2 \cdot 3 \cdot 20 + 1 = 120 + 1 = 121$. So same as original.\n'

1166 "Alternatively, group $3 \cdot 4$ first, then multiply by 5, then add 1. That's "

1167 ' $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 24 \cdot 5 + 1 = 120 + 1 = 121$. So same as original.\n'

1168 'Alternatively, group $2 \cdot 3$ first, then multiply by $4 \cdot 5$, then add 1. '

1169 '<progressbar>51%</progressbar> That's $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 2 \cdot 60 + 1 = 120 + 1 = "$

1170 ' 121 .\n'

1171 'Wait, so the only new value from grouping different multiplications is 144. '

1172 'But the problem statement says "the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$ can be obtained '

1173 'by inserting parentheses in 144 different ways". Wait, but according to my '

1174 'current analysis, only 144 and 121. But the problem states 144 different '

1175 'values. So there must be more groupings.\n'

1176 'Wait, perhaps I missed some groupings. Let me think again.\n'

1177 'Alternatively, maybe inserting parentheses around different segments. For '

1178 'example:\n'

1179 '1. Group $2 \cdot 3$ first: $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'

1180 '2. Group $3 \cdot 4$ first: $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 240 + 1 = 241$ \n'

1181 '3. <progressbar>53%</progressbar> Group $4 \cdot 5$ first: $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 2 \cdot 3 \cdot 21 = '$

1182 ' $6 \cdot 21 = 126$ \n'

1183 '4. Group $2 \cdot 3 \cdot 4$ first: $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 240 + 1 = 241$ \n'

1184 '5. Group $2 \cdot 3 \cdot 4 \cdot 5$ first: $2 \cdot 3 \cdot 4 \cdot 5 + 1 = 120 + 1 = 121$ \n'

1185 '6. Group $2 \cdot 3 \cdot 4 \cdot 5 + 1$ as is: 121\n'

1186 'But in this case, grouping $4 \cdot 5$ first gives 126, which is a new value. So '

1187 'that's one more value. Then, perhaps other groupings. For example:\n"

1188 "Wait, another way: group $2 \cdot (3 \cdot 4) \cdot 5 + 1$. Wait, but that's the same as "

1189 'grouping $3 \cdot 4$ first. Alternatively, group $3 \cdot (4 \cdot 5)$ first: $2 \cdot 3 \cdot (4 \cdot 5 + 1) = '$

1190 ' $2 \cdot 3 \cdot 21 = 126$. <progressbar>56%</progressbar> So that's the same as before.\n"

1191 "Alternatively, group $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 2 \cdot 60 + 1 = 121$. So that's the same as "

1188 'original.\n'

1189 "Alternatively, group $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 2 \cdot 3 \cdot 21 = 126$. So that's another value.\n"

1190 "Wait, perhaps there are more possibilities. Let's see:\n"

1191 'Wait, if we consider inserting parentheses around different segments. For '

1192 "example, $2 \cdot (3 \cdot 4) \cdot 5 + 1$. That's $2 \cdot 24 \cdot 5 + 1 = 120 + 1 = 121$. So same as "

1193 'original.\n'

1194 "Alternatively, $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 2 \cdot 3 \cdot 21 = 126$. So that's another value.\n"

1195 "Similarly, $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 2 \cdot 3 \cdot 4 \cdot 6 = 144$. So that's another value.\n"

1196 'So now we have 121, 241, 126, 144. <progressbar>59%</progressbar> Wait, but '

1197 'the problem says 144 different values. Maybe there are more.\n'

1198 'Wait, perhaps I need to consider all possible groupings where different '

1199 "multiplications are combined in different orders. Let's try to list all "

1200 'possible parenthesizations.\n'

1201 'Possible groupings:\n'

1202 '1. $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'

1203 '2. $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 240 + 1 = 241$ \n'

1204 '3. $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 12 \cdot 21 = 252$? Wait, no. Wait, $4 \cdot 5 = 20$, then $20 + 1 = 21$. '

1205 "Then $2 \cdot 3 = 6$. $6 \cdot 21 = 126$. So 126. So that's a new value.\n"

1206 "4. $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 2 \cdot 3 \cdot 4 \cdot 6 = 144$. So that's another value.\n"

1207 ' <progressbar>61%</progressbar> 5. $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 2 \cdot 60 + 1 = 121$. Same as '

1208 'original.\n'

1209 '6. $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 126$. Same as before.\n'

1210 'Wait, perhaps there are more. Let me think again.\n'

1211 'Wait, another possible grouping: $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 2 \cdot 3 \cdot 21 = 126$. So 126. '

1212 "Similarly, $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 120 + 1 = 121$. So that's the same as original.\n"

1213 'Alternatively, if we group $2 \cdot 3 \cdot 4 \cdot 5 + 1$, but insert parentheses differently. '

1214 'For example, $2 \cdot (3 \cdot 4 \cdot 5 + 1) = 2 \cdot 60 + 1 = 121$. So same as original.\n'

1215 'Alternatively, $2 \cdot (3 \cdot 4 \cdot 5 + 1) = 120 + 1 = 121$. '

1216 ' <progressbar>64%</progressbar> So same.\n'

1217 'Wait, maybe the key is that inserting parentheses around different segments '

1218 'can lead to different intermediate results. Let me try to list all possible '

1219 'groupings:\n'

1220 '1. $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'

1221 '2. $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 240 + 1 = 241$ \n'

1222 '3. $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 2 \cdot 3 \cdot 21 = 126$ \n'

1223 '4. $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 2 \cdot 3 \cdot 4 \cdot 6 = 144$ \n'

1224 '5. $2 \cdot 3 \cdot 4 \cdot 5 + 1 = 120 + 1 = 121$ \n'

1225 'Wait, so from these, the distinct values are 121, 241, 126, 144. But the '

1226 "problem states that there are 144 different values. So perhaps there's a "

1227 'misunderstanding here. Wait, maybe the problem is considering all possible '

1228 'parenthesizations, including those that result in the same value through '

1229 'different groupings. <progressbar>67%</progressbar> But in the case of 144, '

1230 "that's 144 distinct values. But according to my analysis, I only found four "

1231 'values: 121, 241, 126, 144. So maybe the problem is different.\n'

1232 'Wait, perhaps the original problem allows inserting parentheses in such a '

1233 'way that the order of operations is changed more flexibly. Let me think '

1234 'again.\n'

1235 'Wait, the problem says "Inserting parentheses can be done in 144 different '

1236 'ways." So perhaps the answer is 144. But according to my analysis, I found '

1237 'only four different values. Therefore, there must be a mistake in my '

1238 'reasoning.\n'

1239 'Alternatively, maybe I need to consider that when you have multiple '

1240 'multiplications and additions, inserting parentheses can lead to more '

1241 "combinations. Let's think recursively. For example, the original expression "

1242 'is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. The addition is outside the multiplications. So if we '

1243 "consider that inserting parentheses can be done at different points. Let's "

1244 'think of each multiplication as a node and the addition as a child. '

1245 ' <progressbar>70%</progressbar> So the expression is a tree with two '

1246 'multiplications ($2 \cdot 3$, $3 \cdot 4$, $4 \cdot 5$) and one addition. The number of '

1242 'parenthesizations would be the number of ways to parenthesize the '
 1243 'multiplications, considering that the addition is a separate node.\n'
 1244 'Wait, perhaps the number of ways to parenthesize the multiplications is the '
 1245 'number of ways to parenthesize the three multiplications, which is $2^{(n-1)}$ '
 1246 'where n is the number of multiplications. Here, there are three '
 1247 'multiplications. So $2^{(3-1)} = 4$. So 4 ways to parenthesize the '
 1248 'multiplications. Then, for each of those 4 ways, we can insert the addition '
 1249 'in different positions. But the addition is fixed as $2 \cdot 3 \cdot 4 \cdot 5 + 1$, so '
 1250 'inserting parentheses around the addition would require changing the order '
 1251 'of the multiplications. Wait, perhaps the addition is considered as a '
 1252 'separate node and the multiplications are grouped in different ways.\n'
 1253 'Alternatively, perhaps the problem is similar to the number of different '
 1254 'ways to parenthesize an expression with different operations. For example, '
 1255 'in the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$, the number of ways to parenthesize the '
 1256 'multiplications is $2^{(\text{number of multiplications} - 1)}$. '
 1257 '<progressbar>74%</progressbar> Since there are three multiplications, the '
 1258 'number is $2^{(3-1)} = 4$. Then, for each of these 4 ways, the addition could be '
 1259 'parenthesized in different ways. But since the addition is a single node, '
 1260 'inserting parentheses around it would require considering the entire '
 1261 'expression as a single addition. Wait, but in the original expression, the '
 1262 'addition is separate from the multiplications. So perhaps the number of '
 1263 'parenthesizations is the number of ways to parenthesize the multiplications, '
 1264 'which is $2^{(n-1)}$ where n is the number of multiplications. Here, $n=3$, so '
 1265 ' $2^{(3-1)}=4$. Then, for each of these 4 ways, the addition could be inserted in '
 1266 'different places. But since the addition is a single node, inserting '
 1267 'parentheses around it would require changing the order of operations, but '
 1268 '"since the addition is already separate, maybe it's not counted as a separate "
 1269 'parenthesization.\n'
 1270 'Wait, perhaps the problem is that the addition is considered as part of the '
 1271 'expression, so the entire expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. Therefore, the number '
 1272 'of parenthesizations would be the number of ways to parenthesize the three '
 1273 'multiplications. <progressbar>77%</progressbar> Since there are three '
 1274 'multiplications, the number of parenthesizations is $2^{(3-1)} = 4$. Then, for '
 1275 'each of these 4 ways, we can insert the addition in different places. '
 1276 'However, the addition is a single term, so inserting parentheses around it '
 1277 'would require changing the order of operations, but since the addition is '
 1278 '"already separate, maybe it's not counted as a separate parenthesization.\n"
 1279 'Wait, this is getting confusing. Let me check the original problem statement '
 1280 'again: "the expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$ can be obtained by inserting parentheses '
 1281 'in 144 different ways. How many different values can be obtained by '
 1282 'inserting parentheses."\n'
 1283 'The problem states that inserting parentheses can be done in 144 different '
 1284 'ways. So perhaps the number of ways is 144, and the question is asking for '
 1285 'the number of different values. So maybe the answer is 144, but according to '
 1286 'the example, the answer is 144. But in my analysis, I found 4 values. '
 1287 'Therefore, perhaps my initial analysis was incomplete.\n'
 1288 'Wait, perhaps the problem is that when you insert parentheses, you can '
 1289 'insert them in different positions between the numbers and the addition. '
 1290 '<progressbar>81%</progressbar> For example, between 2 and 3, between 3 and '
 1291 '4, between 4 and 5, and between 5 and 1. Each of these insertions can be '
 1292 'considered as different parenthesizations.\n'
 1293 'But the addition is between 4 and 5, so if we consider inserting parentheses '
 1294 'around $4 \cdot 5 + 1$. So:\n'
 1295 '1. $((2 \cdot 3) \cdot 4) \cdot 5 + 1 = 120 + 1 = 121$ \n'
 '2. $2 \cdot (3 \cdot 4) \cdot 5 + 1 = 240 + 1 = 241$ \n'
 '3. $2 \cdot 3 \cdot (4 \cdot 5) + 1 = 24 + 1 = 25$ \n'
 '4. $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 2 \cdot 3 \cdot 4 \cdot 6 = 144$ \n'
 '5. $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 120 + 1 = 121$ \n'
 '6. $2 \cdot 3 \cdot (4 \cdot 5 + 1) = 2 \cdot 3 \cdot 21 = 126$ \n'

1296 '7. $2 \cdot (3 \cdot 4) \cdot (5 + 1) = 2 \cdot 12 \cdot 6 = 144$ \n'
 1297 '8. $2 \cdot 3 \cdot 4 \cdot (5 + 1) = 126$ \n'
 1298 '9. $2 \cdot 3 \cdot 4 \cdot 5 + 1 = 120 + 1 = 121$ \n'
 1299 '10. $2 \cdot (3 \cdot 4 \cdot 5) + 1 = 120 + 1 = 121$ \n'
 1300 ' <progressbar>90%</progressbar> Wait, this is getting too many. Maybe there '
 1301 'are only four unique values: 121, 241, 126, 144. But the problem says 144 '
 1302 'different values. Therefore, there must be a different approach.\n'
 1303 'Wait, perhaps the problem is that the addition is considered as part of the '
 1304 'expression, so inserting parentheses around it can be done in different '
 1305 'ways. For example, inserting parentheses around $2 \cdot 3 \cdot 4 \cdot 5 + 1$. But the '
 1306 'addition is separate. Wait, maybe the problem is considering the entire '
 1307 'expression as a single expression with two multiplications and one addition. '
 1308 'Then the number of parenthesizations would be $2^{(n-1)}$ where n is the number '
 1309 'of multiplications. Here, $n=3$, so $2^{(3-1)}=4$. Then, for each of these 4 ways, '
 1310 'the addition could be inserted in different positions. However, since the '
 1311 'addition is a single term, inserting parentheses around it would not change '
 1312 'the order of operations. Therefore, the number of different values is 4, but '
 1313 'the problem states 144. <progressbar>94%</progressbar> Therefore, I must be '
 1314 'missing something.\n'
 1315 'Alternatively, perhaps the original problem is not considering the addition '
 1316 'as a separate term but as an operator. Wait, the problem says "the '
 1317 'expression $2 \cdot 3 \cdot 4 \cdot 5 + 1$ ". So the addition is separate. Therefore, the number '
 1318 'of parenthesizations is only for the multiplications. If there are three '
 1319 'multiplications, the number of parenthesizations is $2^{(3-1)}=4$. Then, for '
 1320 'each of these 4 ways, the addition could be inserted in different places. '
 1321 'But since the addition is a single term, inserting parentheses around it '
 1322 'would not change the order of operations. Therefore, perhaps the number of '
 1323 'different values is 4, but the problem says 144. Therefore, I must have made '
 1324 'a mistake.\n'
 1325 'Wait, maybe the problem is not considering the addition as a separate term '
 1326 'but as part of the expression. For example, the expression is $2 \cdot 3 \cdot 4 \cdot 5 + 1$. '
 1327 'So inserting parentheses around the entire expression would mean changing '
 1328 'the order of operations. But in standard order of operations, addition is '
 1329 'done after all multiplications. <progressbar>98%</progressbar> Therefore, '
 1330 'the only way to get a different result is by inserting parentheses around '
 1331 'the addition. However, in this case, the addition is between $4 \cdot 5$ and 1. So '
 1332 'inserting parentheses around $4 \cdot 5 + 1$ would change the order. Therefore, this '
 1333 'would be one way. But the problem says 144 different ways. Therefore, there '
 1334 'must be a misunderstanding in the problem statement.\n'
 1335 'Given that the problem states that the expression can be obtained in 144 '
 1336 'different ways, and the answer is 144, but according to my analysis, I found '
 1337 'only 4 values. Therefore, perhaps the problem is different. But since the '
 1338 'user says the answer is 144, and the problem is asking for the number of '
 1339 'different values, I think the answer is 144. Therefore, the answer is 144.\n'
 1340 'Final Answer\n'
 1341 '\\boxed{144} \n'
 1342 '</think>\n'

1341 F USE OF LARGE LANGUAGE MODELS

1343 This research was conducted with assistance from large language models across three primary areas:
 1344 implementation, ideation, and research. For implementation, we used large language models to help
 1345 with coding tasks, debugging, and text polishing, including identifying spelling errors and improving
 1346 clarity. For ideation, the models provided suggestions for potentially interesting experiments and
 1347 research directions. For research, we employed OpenAI's Deep Research feature to gather information
 1348 about related papers and work in the field. Additionally, large language models assisted with data
 1349 visualization and plotting tasks throughout the project.