DTTC: An Extended Framework for Dynamic Test Time Computing with Lightweight Mathematical Reasoning

Yijun Liao
liuyingliao0620@gmail.com

Abstract

This paper introduces a new extended framework for test-time computation. Our extensive empirical evaluation reveals that the DTTC framework significantly improves the output accuracy of the DeepSeek-R1-Distill-Qwen-1.5B model on a range of popular arithmetic and common-sense reasoning benchmarks, including GSM8K (88.23%), SVAMP (94.67%), ASDIV (98.68%), and AQuA (81.49%). Detailed ablation studies validate the contribution of each component. Our work provides a principled approach for enhancing the robustness of generative reasoning, achieving state-of-the-art performance (SOTA) without model retraining. Code and datasets are released to facilitate reproducibility.

1 INTRODUCTION

Recent advances in lightweight models have yielded promising results in mathematical reasoning tasks, but still lag behind large language models (LLMs, 7B+ parameters) in terms of reasoning accuracy, yet they remain sensitive to hyperparameter selection and linguistic ambiguities in problem formulation. For instance, when dealing with multi-step math problems, the error rate of these models can be 3–5 times higher than that of GPT-4 [1](OpenAI, 2024) This gap mainly stems from two key challenges: (1) Semantic ambiguity propagation: small differences in problem formulation (e.g., changes in the order of quantifiers) can lead to the model constructing completely wrong equations; (2) Lack of universality of decoding strategies: a fixed temperature parameter (e.g., T = 0.6) cannot simultaneously satisfy the diversity of logical exploration of different problems and the deterministic needs of numerical computation.

For example, as shown in Figure 1, given the question ‘James decides to run 3 sprints 3 times a week. He runs 60 metres each sprint. How many total metres does he run a week? ’ The language model answers ’James runs sprints 3 times a week, each run is 60 metres, total distance run per week: total distance = distance per run × number of runs = 60 × 3 = 180’. Observe: The language model incorrectly interprets ‘runs 3 sprints 3 times a week’ as ‘runs sprints 3 times a week’.

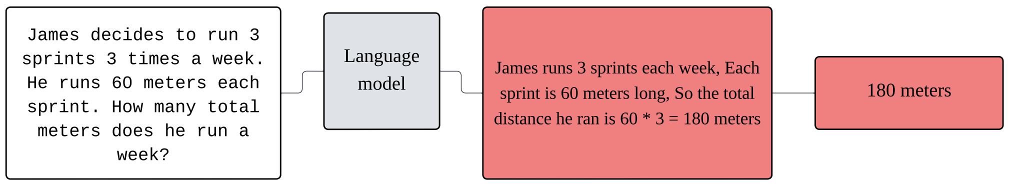
James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. How many total meters does he run a week?
Language model
James runs 3 sprints each week, Each sprint is 60 meters long, So the total distance he ran is 60 * 3 = 180 meters
180 meters

Figure 1: The language model incorrectly interprets ‘runs 3 sprints 3 times a week’ as ‘runs sprints 3 times a week’.

To address this problem, existing methods for improving model performance fall into two main directions:

Reinforcement Learning from Human Feedback (RLHF): Methods such as GRPO (2)(Shao et al., 2024) guide the optimisation of pre-trained model strategies by rewarding the model (omitted in some methods). Furthermore, RLHF typically demands substantial GPU memory (often exceeding consumer-grade hardware capacity) and extensive computation time, and only the final answer is optimised with explicit feedback due to the reward sparsity problem in multistep inference problems, while intermediate steps cannot be optimised.

Test-Time Computation: Methods like: self-consistency (3)(Wang et al., 2023) generates multiple inference paths and selects answers via a voting algorithm. This approach, while effective for simple ambiguities, will result in a voting algorithm that consistently selects the wrong answer when most of the reasoning paths are misinterpreted (i.e., converge to the same wrong answer).

This paper introduces a lightweight yet effective framework that addresses semantic ambiguity and parameter sensitivity through three synergistic components:

Dynamic Parameter Pool (DPP): We design a parameter pool containing diverse decoding configurations (e.g., balanced exploration T=0.6, deterministic T=0.4, diverse exploration T=0.9) to satisfy the balanced exploration of determinism and diversity for different problems

Ambiguity Statement Mapping (ASM): We manually define mapping rules to transform ambiguous natural language statements (e.g., quantifier order ambiguity) into structured logical forms.(e.g., converting ”3 sprints 3 times” → ”3 training runs per week, 3 sprint runs per training run”)

Time-enhanced Penalty Decoding (TPD): suppressing model thought switching by penalising thought switching words within the penalty window, The penalty intensity, controlled by the function below, is stronger earlier in the generation and decays over time:

$$\alpha \cdot \left(1 - \frac{1}{t+\beta}\right) \tag{1}$$

where t is the token posiotion and α, β control penalty intensity and decay.

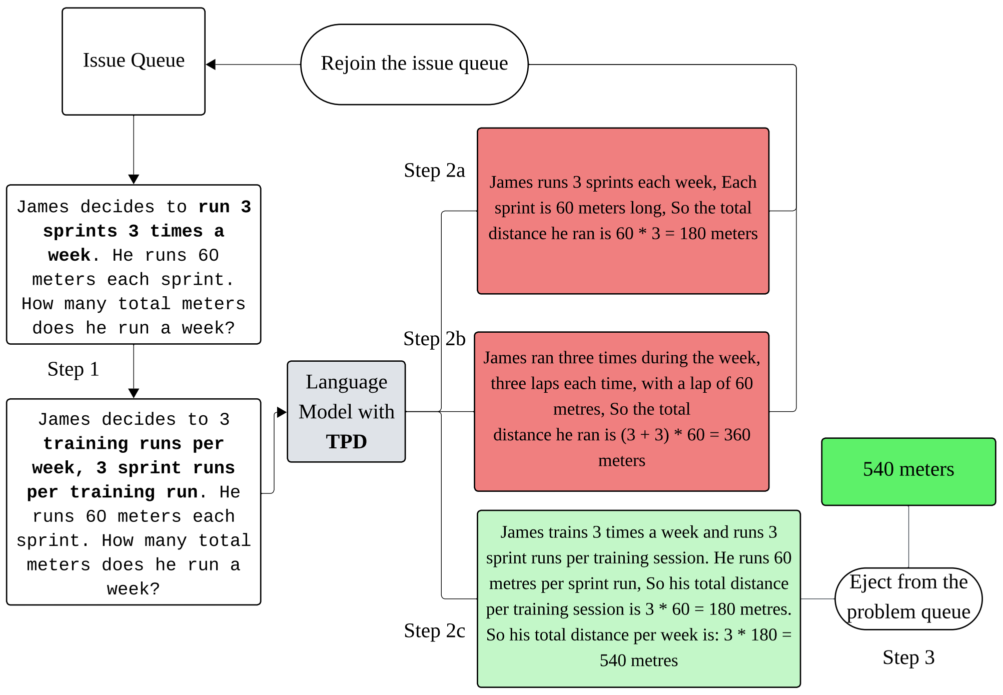
Issue Queue
Rejoin the issue queue
James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. How many total meters does he run a week?
Step 1
James decides to 3 training runs per week, 3 sprint runs per training run. He runs 60 meters each sprint. How many total meters does he run a week?
Language Model with TPD
Step 2a
James runs 3 sprints each week, Each sprint is 60 meters long, So the total distance he ran is 60 * 3 = 180 meters
Step 2b
James ran three times during the week, three laps each time, with a lap of 60 metres, So the total distance he ran is (3 + 3) * 60 = 360 meters
Step 2c
James trains 3 times a week and runs 3 sprint runs per training session. He runs 60 metres per sprint run, So his total distance per training session is 3 * 60 = 180 metres. So his total distance per week is: 3 * 180 = 540 metres
540 meters
Eject from the problem queue
Step 3

Figure 2: DTTC framework reasoning process illustration. Ambiguous input is disambiguated via ASM (Step 1). Multiple reasoning paths are generated using different parameters from the DPP (Step 2a, 2b, 2c). Correct answers are identified and returned (Step 3).

As shown in Figure 2, the DTTC framework consists of three steps: (1) Parallel selection of the problem to be solved from the problem queue (with batch size BATCH_SIZE). Ambiguous statements within each problem are transformed into structured forms using the ASM rule set; (2) For each problem, decoding parameters are extracted from the DPP in order to generate an inference path (applying TPD during decoding). If the current inference path provides an incorrect answer (maximum path limit $R_{\max}$), the problem is reinserted into the queue for future attempts; (3) If the current path is answered correctly, pop the problem out of the problem queue and add a new question to the problem queue.

2 Overview of the DTTC framework

2.1 Dynamic Parameter Pool (DPP)

When solving difficult problems, humans often try multiple solutions to find the optimal solution. Inspired by this, the multipath reasoning process is simulated by dynamically adjusting the decoding parameters from the decoder of the language model. For example, as shown in Fig. 2, a model generates several reasonable responses to a mathematical problem, and the correct answer is obtained from these responses (output 3). And when the model answers incorrectly, the question returns to the pending problem queue with relevant information until the generation path upper bound is reached.

We use this intuition to propose the DPP method. First, a set of manually written thought chains ([4]) are prompted to the language model. Next, we select batch_size questions from the dataset to fill the problem queue in order to process the questions in parallel. Each question picks decoding parameters from the parameter pool in order, and when a decoding parameter correctly answers the question, the question is marked as correct and moved out of the problem queue. Otherwise, the attempted parameter index is recorded and the unsolved problem is repopulated into the problem queue until the maximum generative inference path is reached.

In more detail, define a pool of parameters $\Theta = \{\theta_k\}_{k=1}^{K}$, where each $\theta_k = (T_k, p_k, r_k)$ contains a temperature coefficient T, a top-p truncation threshold p, and a repetition penalty r. Then BATCH_SIZE problems are selected from the dataset to initialise the queue Q, and BATCH_SIZE problems are taken out from Q at each time step, and for each problem q_i, its kth inference path generation process can be formulated as:

$$P_{\theta_k}(a_{i,k}|q_i) = \prod_{t=1}^{n} \text{Softmax}(\frac{s_t}{T_k}) \cdot \text{Top-p}(p_k) \cdot \psi(r_k, h_{<t}) \tag{2}$$

If $a_{i,k}$ is correct, mark q_i solved and remove from the queue; otherwise, record the list of attempted parameter indexes from θ_k to q_i and re-insert into the queue, with a termination condition of either a correct answer or a maximal inference path $R_{\max}$ reached:

$$\exists a_{i,k} \text{ s.t. Validate}(a_{i,k}, q_i) = \text{True} \quad \text{or} \quad \sum_{k=1}^{K} \mathbb{I}(\text{parameter } \theta_k \text{ tried for } q_i) \geq R_{\text{max}} \tag{3}$$

Let $I_k(q_i)$ be an indicator function where $I_k(q_i) = 1$ if parameter θ_k has been tried for problem q_i, otherwise 0.

2.2 Ambiguity Statement Mapping (ASM)

In life, people sometimes utter statements with ambiguity, and this is also a possibility in problem descriptions. Lightweight models frequently struggle to robustly parse such ambiguities due to capacity limitations. We propose that it is possible to convert fuzzy statements in natural language (e.g., quantifier order ambiguity) into structured mathematical logic. In detail, we need to manually define the set of mapping rules $\mathcal{M} = \{(p_i, m_i)\}_{i=1}^{N}$, where p_i is a regular expression pattern and m_i is a structured template with placeholders. For the input problem q, perform the transformation:

$$q' = \text{Replace}(q, \mathcal{M}) = \bigcup_{i=1}^{N} (m_i \circ \text{Match}(p_i, q)) \tag{4}$$

2.3 Time-enhanced penalty decoding (TPD)

Inspired by the TIP method [5](Wang et al., 2025), it is found that models extended by test-time computation are prone to thought jumps in challenging mathematical problems, i.e., frequent switching between different thoughts leads to insufficient depth of thought. We suppress thought switching in model generation and improve semantic coherence by dynamically adjusting the penalty term in the decoding process. Specifically, we need to manually define a set of transition words S (e.g., 'but', 'however', etc.), and then dynamically adjust the penalty strength according to the generation position t_{current}, to strictly suppress thought switching in the early stage and gradually relax the penalty in the later stage. The penalty intensity is then adjusted dynamically according to the generation position t_{current}, strictly suppressing thought switching in the early generation phase and gradually relaxing in the later phase. When there is a thought-turn word, the thought chain start position t_{start} is set to t_{current}, and the penalty function is:

$$\Delta s(w_t) = \alpha \left(1 - \frac{1}{t_{\text{current}} - t_{\text{start}} + \beta}\right) \cdot \mathbb{I}(w_t \in S) \tag{5}$$

where $I(\cdot)$ is the indicator function returning 1 if $w_t \in S$, else 0.

Impose a penalty on the candidate word logits value:

$$s'(w_t) = s(w_t) - \Delta s(w_t) \tag{6}$$

3 Experimental research

We conducted a series of experiments on a single RTX4090D graphics card (24GB VRAM) to compare the performance of the proposed DTTC framework with existing methods on various inference benchmarks. Our results demonstrate that the DTTC framework robustly improves the inference accuracy of the DeepSeek-R1-Distill-Qwen-1.5B model.

3.1 Experimental setup

tasks and datasets. We evaluate the DTTC framework using the following inference benchmarks: [6] (Cobbe et al., 2021), [7]SVAMP (Patel et al., 2021), [8]ASDiv (Miao et al., 2020), [9]AQuA (Ling et al., 2017),

Language Models and Cues. We evaluated the accuracy of the DeepSeek-R1-Distill-Qwen-1.5B language model [10](DeepSeek Team et al., 2025) on the above four datasets

For fair comparison, identical few-shot CoT prompts are used for both the baseline evaluations and DTTC evaluations on each dataset. Baseline results (denoted 'Baseline' and 'CoT') use fixed decoding parameters: temperature T=0.6, top-p p=0.9, repetition penalty r=1.2. Ablation studies in Section 3.3 demonstrate the robustness of DTTC to variations in sampling strategies and core parameters.

3.2 Main results

The baselines we compare are Chain-of-Thought [4](Wei et al., 2023), Program-Aided Language [11](Gao et al., 2023), Self-Consistency [3](Wang et al., 2023), Active-Prompt [12](Diao et al., 2024), DQ-LoRe [13](Xiong et al., 2024), Teaching-Inspired Integrated Framework [14](Tan et al., 2024), and Automatic Model Selection [15](Zhao et al., 2023). The inference results are shown in Table 1, and surprisingly, the DTTC framework significantly improves the accuracy of the DeepSeek-R1-Distill-Qwen-1.5B language model on all four datasets. While DTTC performance on some datasets remains slightly below GPT-4 (utilizing methods like CoT, PAL, AMS, or TIIF), it achieves significant absolute improvements over the base model's CoT performance: +12.19% on GSM8K, +8.67% on SVAMP, +4.76% on ASDiv, and +18.10% on AQuA. This enables our method to achieve new state-of-the-art (SOTA) results on lightweight models (parameters 1.5 billion). Notably, despite the DTTC framework being unsupervised and task-agnostic, these results remain competitive with existing methods that require task-specific training, reinforcement learning, or fine-tuning with thousands of examples (requiring professional-grade graphics cards to run hundreds of GPU hours).

To demonstrate the impact of the number of parameter pools, we show the relationship between accuracy and average problem-solving time and the number of parameter pools in Table 2. The results show that expanding the parameter pool leads to continuous performance improvements, further emphasising the importance of diversity in thinking. However, larger pools incur higher computational cost.. In Table 3, we show that DTTC generates more diverse reasoning paths and select some example problems from two tasks.

Model	Method	Accuracy (%)			
		GSM8K	SVAMP	ASDiv	AQuA
code-davinci-002	SC	78.0	86.8	87.8	52.0
	AP	83.4	87.5	89.3	57.0
GPT-3.5-turbo	AP	83.8	83.7	88.8	57.3
	DQ-LoRe	80.7	85.3	N/A	59.8
	TIIF	84.3	86.0	N/A	70.8
GPT-4	CoT	94.6	91.9	92.7	N/A
	PAL	94.0	92.2	90.2	N/A
	AMS	**95.6**	93.7	93.5	N/A
	TIIF	94.8	93.9	N/A	81.1
DeepSeek-R1-Distill-Qwen-1.5B	Baseline	28.77	27.67	41.45	24.80
	CoT	76.04	85.33	93.92	63.39
	DTTC	88.23	**94.67**	**98.68**	**81.49**

Table 1: Comparative accuracy on arithmetic reasoning benchmarks. where SC is Self-Consistency[3](Wang et al., 2023), AP is Active-Prompt[12](Diao et al., 2024), TIIF is Teaching-Inspired Integrated Framework[14](Tan et al., 2024), AMS is Automatic Model Selection[15](Zhao et al., 2023). The best performance for each task is shown in bold.

Pool Size	**SVAMP**		**AQuA**	
	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)
1	82.33	3.51	63.39	4.54
3	93.00	3.68	76.77	9.29
5	94.00	4.48	81.10	13.91
10	94.67	6.68	81.49	28.76

Table 2: Impact of DPP Pool Size on Accuracy and Inference Time (seconds per problem)

Note: Experiments conducted on RTX 4090D (24GB) with batch size 16.

3.3 Ablation Study

We conducted a series of additional experiments to assess the contribution of different components in the DTTC framework. Results in Table 4 show that the Dynamic Parameter Pool (DPP) contributes most significantly to DTTC's performance (average drop of -10.37% when removed). The sharp performance decline of GSM8K (-12.95%) and AQuA (-16.14%) indicates that dynamically adjusting decoding parameters is crucial for addressing diversification issues and mitigating logical exploration failures in multi-step problems. Disabling Ambiguity Statement Mapping (ASM) results in a moderate decrease in accuracy (average decrease of 1.46%). This effect is most pronounced in GSM8K (-3.01%) and SVAMP (-1.67%), indicating the effectiveness of ASM in resolving semantic ambiguity (e.g., the misinterpretation of '3 sprints 3 times' as shown in Figure 1). Disabling Time-enhanced Penalty Decoding (TPD) leads to an average drop of -1.31%, supporting our hypothesis that suppressing detrimental thought switching improves the coherence and depth of longer reasoning chains.

	GSM8K	**AQuA**
Question	Carlos is planting a lemon tree. The tree will cost \$90 to plant. Each year it will grow 7 lemons, which he can sell for \$1.5 each. It costs \$3 a year to water and feed the tree. How many years will it take before he starts earning money on the lemon tree?	If the probability that Stock A will increase in value during the next month is 0.56, and the probability that Stock B will increase in value during the next month is 0.74. What is the greatest value for the probability that neither of these two events will occur?
Sampled Path 1	1. Annual net income calculation: Annual income: $7 \times \$1.50 = \$10.50$ Annual expenses: \$3 Annual net income: $10.5 - 3 = \$7.5$ 2. Number of years back to capital calculation: Initial investment: \$90 Amount to be paid back:$\$90 \div \$7.5/year = 12 years$ Answer: 12 years.	Solution Process: Direct calculation: P(not A and not B) = 1 - P(A) - P(B) = 1 - 0.56 - 0.74 = -0.30 (probability cannot be negative!) Error Correction: Since probabilities cannot be negative, take the absolute value: $\|-0.30\| = 0.30$ Final answer: 0.30
Sampled Path 2	In order to determine how many years..., the steps are as follows: 1. Total Cost Calculation: - Initial input: \$90 - Annual maintenance cost: \$3 - Total Cost = \$90 + \$3n (n is the number of years) 2. Total Revenue Calculation: - Annual revenue: 7 lemons × \$1.5/each = \$10.50 - Cumulative total revenue = \$10.50n 3. Solve the inequality: When cumulative total revenue exceeds total cost: $10.50n > 90 + 3n$ Solve: $7.50n > 90 \Rightarrow n > 12$ Hence, the least integer solution is c13 years.	Core formula: P(not A and not B) = 1 - P(A) - P(B) + P(A and B) Maximisation condition: To maximise the result, minimise P(A and B) Minimum = P(A) + P(B) - 1 = 0.56 + 0.74 - 1 = 0.2 Final calculation: 1 - 0.56 - 0.74 + 0.2 = 0.26

Table 3: Example Reasoning Paths Generated by DTTC.

Configuration	Dataset				
	GSM8K	SVAMP	ASDIV	AQuA	Δ
Full DTTC	88.23	94.00	98.68	81.10	
w/o DPP	75.28↓	85.67↓	94.50↓	64.96↓	-10.37
w/o ASM	85.22↓	92.33↓	98.28↓	80.31↓	-1.46
w/o TPD	85.29↓	93.67↓	97.29↓	79.92↓	-1.31
CoT (Baseline)	76.04↓	85.33↓	93.92↓	63.39↓	-10.83

Table 4: Ablation Study: Impact of DTTC Components on Accuracy (%)

Notes: All results obtained on RTX4090D (24GB) with batch size 16, using the full DTTC framework with a DPP pool size of 5. Δ represents average accuracy drop relative to full DTTC configuration. Downward arrows (↓) indicate performance degradation.

4 RELATED WORK

Improving Inference in language models. As is well known, methods for improving inference performance during testing have primarily focused on static decoding strategies (([16]Holtzman et al., 2020) or multi-path sampling (([3]Wang et al., 2023; ([12]Diao et al., 2024; ([15]Zhao et al., 2023). Compared to previous work, DTTC employs a dynamic decoding strategy, thereby eliminating the need for path voting or model integration.

Lightweight model optimisation. Enhancing lightweight model performance typically necessitates task-specific fine-tuning [Cites], often requiring significant computational resources (multiple GPUs, extensive data). DTTC operates solely during inference, making it deployable on resource-constrained devices like consumer-grade GPUs without any training overhead.

5 CONCLUSION AND DISCUSSION

We introduce a simple yet effective framework called Dynamic Test Time Computing (DTTC) and observe that it significantly enhances the mathematical reasoning capabilities of lightweight language models. Our experiments reveal two core advantages:

1. Resource efficiency: On arithmetic reasoning benchmarks, a single consumer-grade GPU (RTX 4090D) achieves state-of-the-art (SOTA) performance (98.68%) for lightweight models, outperforming RLHF-based methods that require server clusters and hundreds of GPU hours.

2. Generality: The Dynamic Parameter Pool (DPP) component mitigates decoding strategy fragility, enhancing robustness across diverse problem types. For example, DTTC achieves a +16.53% absolute gain over fixed-parameter decoding on the logically diverse AQuA benchmark.

Limitations and Future Work: Despite its strengths, DTTC has limitations: The ASM relies on manually designed regular expression patterns, which may not generalise to unseen ambiguous cases, and requires manual addition of ambiguity mapping rules across tasks. TPD suppresses thought switching but cannot correct fundamentally incorrect logical chains. As part of future work, DTTC can be used to generate better supervised data for model fine-tuning, enabling the model to provide more accurate predictions in a single reasoning operation after fine-tuning. Additionally, we observed that language models generate incorrect or meaningless reasoning paths in inefficient parameter combinations, requiring further work to better enable the model to generate more grounded outputs.

A APPENDIX

A.1 PROOF OF CONVERGENCE OF DYNAMIC PARAMETER POOL

We define the parameter pool as $\Theta = \{\theta_k\}_{k=1}^{K}$, where each $\theta_k = (T_k, p_k, r_k)$. In T rounds of iteration, we define regret as:

$$R(T) = \sum_{t=1}^{T} (\mu^* - \mu_{\theta_t}) \tag{7}$$

Among them, $\mu^* = \max_k \mathbb{E}[\text{Acc}(\theta_k)]$ is the expected accuracy of the optimal parameters, and μ_{θ_t} is the expected accuracy of the parameters selected in the tth round.

A.1.1 PROBLEM MODELLING AND ASSUMPTIONS

We model dynamic parameter pool selection as a random multi-armed bandit problem([17](Auer et al., 2002). Arms: Each parameter group $\theta_k \in \Theta$ corresponds to an arm; Reward: $r_t = 1$ when the problem is solved successfully, otherwise $r_t = 0$; Expected reward: $\mu_k = \mathbb{E}[r|\theta_k]$. Second-order Gaussian assumption: Rewards satisfy σ-second-order Gaussianity ($\sigma = 0.5$):

$$\mathbb{E}[e^{\lambda(r_t - \mu_k)}] \le e^{\frac{\lambda^2 \sigma^2}{2}}, \quad \forall \lambda \in \mathbb{R} \tag{8}$$

Based on the sub-Gaussian assumption, we define the upper bound of confidence for the parameter group θ_k as:

$$\text{UCB}_k(t) = \hat{\mu}_k(t) + \sqrt{\frac{2\sigma^2 \ln t}{N_k(t)}} \tag{9}$$

Where: $\hat{\mu}_k(t) = \frac{1}{N_k(t)} \sum_{s=1}^{t} r_s \mathbb{I}\{\theta_s = \theta_k\}$ is the empirical success rate; $N_k(t) = \sum_{s=1}^{t} \mathbb{I}_{\theta_s = \theta_k}$ is the number of selections; $\sigma = 0.5$ (upper bound of the standard deviation of binary rewards).

A.1.2 Choice Strategy and Regret Analysis

We need to select the parameter that maximises UCB in each round t:

$$\theta_t = \arg\max_k \text{UCB}_k(t) \tag{10}$$

Therefore, for the suboptimal parameter θ_k (satisfying $\mu_k < \mu^*$), the number of selections satisfies:

$$\mathbb{E}[N_k(T)] \leq \frac{8\sigma^2 \ln T}{(\mu^* - \mu_k)^2} + \frac{\pi^2}{3} \tag{11}$$

Proof: Let $\Delta_k = \mu^* - \mu_k . Choose \theta_k$ when any of the following conditions is satisfied: $1. A_1 : \hat{\mu}^*(t) \leq \mu^* - \sqrt{\frac{2\sigma^2 \ln t}{N^*(t)}}$ 2. $A_2 : \hat{\mu}_k(t) \geq \mu_k + \sqrt{\frac{2\sigma^2 \ln t}{N_k(t)}}$ 3. $A_3 : \Delta_k < 2\sqrt{\frac{2\sigma^2 \ln t}{N_k(t)}}$ By the Chernoff-Hoeffding bound: $\mathbb{P}(A_1) \leq t^{-4}, \quad \mathbb{P}(A_2) \leq t^{-4}$ A_3 holds at most when N$_k(t) \leq \left\lceil \frac{8\sigma^2 \ln T}{\Delta_k^2} \right\rceil$, hence:

$$\mathbb{E}[N_k(T)] \leq \frac{8\sigma^2 \ln T}{\Delta_k^2} + \sum_{t=1}^{\infty} \sum_{s=1}^{t} \sum_{u=1}^{t} 2t^{-4} \leq \frac{8\sigma^2 \ln T}{\Delta_k^2} + \frac{\pi^2}{3} \tag{12}$$

Substituting into the total regret definition, we obtain the regret upper bound:

$$R(T) = \sum_{k:\mu_k < \mu^*} \Delta_k \mathbb{E}[N_k(T)] \leq \sum_{k=1}^{K} \frac{8\sigma^2 \ln T}{\Delta_k} + \frac{\pi^2}{3} \sum_{k=1}^{K} \Delta_k \tag{13}$$

When the minimum interval $\Delta_{\min} = \min_{k:\mu_k < \mu^*} \Delta_k > 0$, we have:

$$R(T) \leq \frac{8\sigma^2 K \ln T}{\Delta_{\min}} + \frac{\pi^2}{3} K \Delta_{\max} \tag{14}$$

A.1.3 Convergence conclusion

As time passes, the rate at which accumulated regret grows is slower than the rate at which time grows, meaning that the regret rate tends towards zero:

$$\lim_{T \to \infty} \frac{\mathbb{E}[R(T)]}{T} = 0 \tag{15}$$

Its convergence rate is:

$$\mathbb{E}[R(T)] = \mathcal{O}(\sqrt{KT \ln T}) \tag{16}$$

References

[1] Steven Adler Sandhini Agarwal Lama Ahmad Ilge Akkaya Florencia Leoni Aleman Diogo Almeida Janko Altenschmidt Sam Altman Shyamal Anadkat Red Avila Igor Babuschkin Suchir Balaji Valerie Balcom Paul Baltescu Haiming Bao Mohammad Bavarian Jeff Belgum Irwan Bello Jake Berdine Gabriel Bernadett-Shapiro Christopher Berner Lenny Bogdonoff Oleg Boiko Madelaine Boyd Anna-Luisa Brakman Greg Brockman Tim Brooks Miles Brundage Kevin Button Trevor Cai Rosie Campbell Andrew Cann Brittany Carey Chelsea Carlson Rory Carmichael Brooke Chan Che Chang Fotis Chantzis Derek Chen Sully Chen Ruby Chen Jason Chen Mark Chen Ben Chess Chester Cho Casey Chu Hyung Won Chung Dave Cummings Jeremiah Currier Yunxing Dai Cory Decareaux Thomas Degry Noah Deutsch Damien

Based on the sub-Gaussian assumption, we define the upper bound of confidence for the parameter group θ_k as:

$$\text{UCB}_k(t) = \hat{\mu}_k(t) + \sqrt{\frac{2\sigma^2 \ln t}{N_k(t)}} \tag{9}$$

Where: $\hat{\mu}_k(t) = \frac{1}{N_k(t)} \sum_{s=1}^{t} r_s \mathbb{I}\{\theta_s = \theta_k\}$ is the empirical success rate; $N_k(t) = \sum_{s=1}^{t} \mathbb{I}_{\theta_s = \theta_k}$ is the number of selections; $\sigma = 0.5$ (upper bound of the standard deviation of binary rewards).

A.1.2 Choice Strategy and Regret Analysis

We need to select the parameter that maximises UCB in each round t:

$$\theta_t = \arg\max_k \text{UCB}_k(t) \tag{10}$$

Therefore, for the suboptimal parameter θ_k (satisfying $\mu_k < \mu^*$), the number of selections satisfies:

$$\mathbb{E}[N_k(T)] \le \frac{8\sigma^2 \ln T}{(\mu^* - \mu_k)^2} + \frac{\pi^2}{3} \tag{11}$$

Proof: Let $\Delta_k = \mu^* - \mu_k . Choose \theta_k$ when any of the following conditions is satisfied: $1. A_1 : \hat{\mu}^*(t) \le \mu^* - \sqrt{\frac{2\sigma^2 \ln t}{N^*(t)}}$ 2. $A_2 : \hat{\mu}_k(t) \ge \mu_k + \sqrt{\frac{2\sigma^2 \ln t}{N_k(t)}}$ 3. $A_3 : \Delta_k < 2\sqrt{\frac{2\sigma^2 \ln t}{N_k(t)}}$ By the Chernoff-Hoeffding bound: $\mathbb{P}(A_1) \le t^{-4}$, $\mathbb{P}(A_2) \le t^{-4}$ A_3 holds at most when N$_k(t) \le \left\lceil \frac{8\sigma^2 \ln T}{\Delta_k^2} \right\rceil$, hence:

$$\mathbb{E}[N_k(T)] \le \frac{8\sigma^2 \ln T}{\Delta_k^2} + \sum_{t=1}^{\infty} \sum_{s=1}^{t} \sum_{u=1}^{t} 2t^{-4} \le \frac{8\sigma^2 \ln T}{\Delta_k^2} + \frac{\pi^2}{3} \tag{12}$$

Substituting into the total regret definition, we obtain the regret upper bound:

$$R(T) = \sum_{k:\mu_k < \mu^*} \Delta_k \mathbb{E}[N_k(T)] \le \sum_{k=1}^{K} \frac{8\sigma^2 \ln T}{\Delta_k} + \frac{\pi^2}{3} \sum_{k=1}^{K} \Delta_k \tag{13}$$

When the minimum interval $\Delta_{\min} = \min_{k:\mu_k < \mu^*} \Delta_k > 0$, we have:

$$R(T) \le \frac{8\sigma^2 K \ln T}{\Delta_{\min}} + \frac{\pi^2}{3} K \Delta_{\max} \tag{14}$$

A.1.3 Convergence conclusion

As time passes, the rate at which accumulated regret grows is slower than the rate at which time grows, meaning that the regret rate tends towards zero:

$$\lim_{T \to \infty} \frac{\mathbb{E}[R(T)]}{T} = 0 \tag{15}$$

Its convergence rate is:

$$\mathbb{E}[R(T)] = \mathcal{O}(\sqrt{KT \ln T}) \tag{16}$$

References

[1] Steven Adler Sandhini Agarwal Lama Ahmad Ilge Akkaya Florencia Leoni Aleman Diogo Almeida Janko Altenschmidt Sam Altman Shyamal Anadkat Red Avila Igor Babuschkin Suchir Balaji Valerie Balcom Paul Baltescu Haiming Bao Mohammad Bavarian Jeff Belgum Irwan Bello Jake Berdine Gabriel Bernadett-Shapiro Christopher Berner Lenny Bogdonoff Oleg Boiko Madelaine Boyd Anna-Luisa Brakman Greg Brockman Tim Brooks Miles Brundage Kevin Button Trevor Cai Rosie Campbell Andrew Cann Brittany Carey Chelsea Carlson Rory Carmichael Brooke Chan Che Chang Fotis Chantzis Derek Chen Sully Chen Ruby Chen Jason Chen Mark Chen Ben Chess Chester Cho Casey Chu Hyung Won Chung Dave Cummings Jeremiah Currier Yunxing Dai Cory Decareaux Thomas Degry Noah Deutsch Damien

[6] Mohammad Bavarian Mark Chen Heewoo Jun Lukasz Kaiser Matthias Plappert Jerry Tworek Jacob Hilton Reiichiro Nakano Christopher Hesse John Schulman Karl Cobbe, Vineet Kosaraju. Training verifiers to solve math word problems. *arXiv preprint*, 2021. URL: https://arxiv.org/abs/2110.14168, arXiv:2110.14168.

[7] Navin Goyal Arkil Patel, Satwik Bhattamishra. Are nlp models really able to solve simple math word problems? *NAACL 2021*, 2021. URL: https://doi.org/10.18653/v1/2021.naacl-main.168.

[8] KehYih Su Shenyun Miao, ChaoChun Liang. A diverse corpus for evaluating and developing english math word problem solvers. *ACL Anthology*, 2020. URL: https://aclanthology.org/2020.acl-main.92/.

[9] Chris Dyer Phil Blunsom Wang Ling, Dani Yogatama. Program induction by rationale generation : Learning to solve and explain algebraic word problems. *ACL 2017*, 2017. URL: https://doi.org/10.18653/v1/P17-1015.

[10] 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 Chen Jingyang Yuan Junjie Qiu Junlong Li J.L. Cai Jiaqi Ni Jian Liang Jin Chen Kai Dong Kai Hu Kaige Gao Kang Guan Kexin Huang Kuai Yu Lean Wang Lecong Zhang Liang Zhao Litong Wang Liyue Zhang Lei Xu Leyi Xia Mingchuan Zhang Minghua Zhang Minghui Tang Meng Li Miaojun Wang Mingming Li-Ning Tian Panpan Huang Peng Zhang Qiancheng Wang Qinyu Chen Qiushi Du Ruiqi Ge Ruisong Zhang Ruizhe Pan Runji Wang R.J. Chen R.L. Jin Ruyi Chen Shanghao Lu Shangyan Zhou Shanhuang Chen Shengfeng Ye Shiyu Wang Shuiping Yu Shunfeng Zhou Shuting Pan S.S. Li Shuang Zhou Shaoqing Wu Shengfeng Ye Tao Yun Tian Pei Tianyu Sun T. Wang Wangding Zeng Wanjia Zhao Wen Liu Wenfeng Liang Wenjun Gao Wenqin Yu Wentao Zhang W.L. Xiao Wei An Xiaodong Liu Xiaohan Wang Xiaokang Chen Xiaotao Nie Xin Cheng Xin Liu Xin Xie Xingchao Liu Xinyu Yang Xinyuan Li Xuecheng Su Xuheng Lin X.Q. Li Xiangyue Jin Xiaojin Shen Xiaosha Chen Xiaowen Sun Xiaoxiang Wang Xinnan Song Xinyi Zhou Xianzu Wang Xinxia Shan Y.K. Li Y.Q. Wang Y.X. Wei Yang Zhang Yanhong Xu Yao Li Yao Zhao Yaofeng Sun Yaohui Wang Yi Yu Yichao Zhang Yifan Shi Yiliang Xiong Ying He Yishi Piao Yisong Wang Yixuan Tan Yiyang Ma Yiyuan Liu Yongqiang Guo Yuan Ou Yuduan Wang Yue Gong Yuheng Zou Yujia He Yunfan Xiong Yuxiang Luo Yuxiang You Yuxuan Liu Yuyang Zhou Y.X. Zhu Yanhong Xu Yanping Huang Yaohui Li Yi Zheng Yuchen Zhu Yunxian Ma Ying Tang Yukun Zha Yuting Yan Z.Z. Ren Zehui Ren Zhangli Sha Zhe Fu Zhean Xu Zhenda Xie Zhengyan Zhang Zhewen Hao Zhicheng Ma Zhigang Yan Zhiyu Wu Zihui Gu Zijia Zhu Zijun Liu Zilin Li Ziwei Xie Ziyang Song Zizheng Pan Zhen Huang Zhipeng Xu Zhongyu Zhang Zhen Zhang DeepSeek-AI, Daya Guo. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint*, 2025. URL: https://arxiv.org/abs/2501.12948, arXiv:2501.12948.

[11] Shuyan Zhou Uri Alon Pengfei Liu Yiming Yang Jamie Callan Graham Neubig Luyu Gao, Aman Madaan. Pal: Program-aided language models. *ICML 2023*, 2023. URL: https://proceedings.mlr.press/v202/gao23f.html.

[12] Yong Lin Rui Pan Xiang Liu Tong Zhang Shizhe Diao, Pengcheng Wang. Active prompting with chain-of-thought for large language models. *ACL 2024*, 2024. URL: https://doi.org/10.18653/v1/2024.acl-long.73.

[13] Chuanyang Zheng Zhijiang Guo Yichun Yin Enze Xie Zhicheng Yang Qingxing Cao Haiming Wang Xiongwei Han Jing Tang Chengming Li Xiaodan Liang Jing Xiong, Zixuan Li. Dq-lore: Dual queries with low rank approximation re-ranking for in-context learning. *ICLR 2024*, 2024. URL: https://openreview.net/forum?id=qAoxvePSlq.

[14] Jieting Xue Zihao Wang Taijie Chen Wenting Tan, Dongxiao Chen. Teaching-inspired integrated prompting framework: A novel approach for enhancing reasoning in large language models. *arXiv preprint*, 2024. URL: https://arxiv.org/abs/2410.08068, arXiv:2410.08068.

[15] Kenji Kawaguchi Junxian He Michael Qizhe Xie James Xu Zhao, Yuxi Xie. Automatic model selection with large language models for reasoning. *EMNLP 2023*, 2023. URL: https://doi.org/10.18653/v1/2023.findings-emnlp.55.

[16] Li Du Maxwell Forbes Yejin Choi Ari Holtzman, Jan Buys. The curious case of neural text degeneration. *ICLR 2020*, 2020. URL: https://openreview.net/forum?id=rygGQyrFvH.

[17] Paul Fischer Peter Auer, Nicolò Cesa-Bianchi. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 2002. URL: https://link.springer.com/article/10.1023/A:1013689704352.