Test-Time Optimization for Small Language Models (SLMs)

- By **Yash Bhaskar, Akanksha Srivastava**

Abstract

Small Language Models (SLMs) often struggle with complex reasoning tasks despite possessing relevant knowledge. This work explores test-time compute allocation as a critical factor influencing reasoning performance, an area currently underexplored for SLMs. We investigate techniques like budget forcing, compare Chain of Thought (CoT) with Chain of Draft (CoD) prompting, and experiment with Supervised Fine-Tuning (SFT) and Group Relative Policy Optimization (GRPO) to enhance SLM reasoning. We benchmark performance on datasets like MATH-500, AIME24, and GPQA, detailing our attempts to replicate and adapt methods from larger models to SLMs (specifically Qwen-2.5-1.5B-Instruct, Llama-3.2-1B-Instruct, and Phi-4-3.8B-mini-Instruct). Our findings highlight the potential of test-time optimization while underscoring the challenges in achieving consistent reasoning format adherence and stability with current SLMs.

1. Introduction

Large Language Models (LLMs) have shown remarkable capabilities, yet they often struggle with complex reasoning problems even when they possess the necessary underlying knowledge [3]. A key factor influencing performance, particularly in reasoning, is the allocation of computational resources at test time. Allowing models more "thinking time" can significantly improve accuracy, but this aspect remains relatively underexplored, especially for Small Language Models (SLMs).

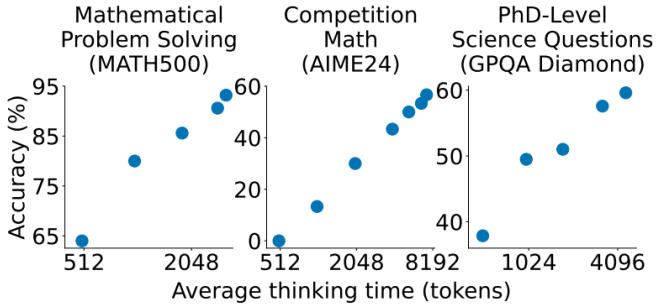


Figure 1. **Test-time scaling with s1-32B.** We benchmark **s1-32B** on reasoning-intensive tasks and vary test-time compute.

Figure 1 illustrates how increasing the average thinking time (measured in tokens) can boost the accuracy of an SLM (s1-32B, as presented in the source material, potentially a baseline LLM or target size) on reasoning-intensive tasks. This motivates the exploration of techniques that explicitly control or optimize test-time compute for smaller, more resource-constrained models. This report investigates several such techniques, focusing on their applicability and effectiveness for SLMs.

2. Related Work

Several recent works have explored optimizing LLM reasoning, particularly concerning test-time computation and efficiency.

1. **Chain of Draft (CoD)** [1] introduced a method aimed at "Thinking Faster by Writing Less". CoD reduces inference time significantly (reportedly by an order of magnitude) compared to standard **Chain of Thought (CoT)** while maintaining comparable performance by generating more concise intermediate reasoning steps.
2. **Thinking-Optimal Scaling** [2] explored the impact of scaling the length of Chain of Thoughts (CoTs) on reasoning performance, demonstrating that longer reasoning traces often correlate with better accuracy, up to a point.
3. **S1: Simple test-time scaling** [3] proposed "budget forcing" as an effective strategy to control the amount of reasoning tokens generated by a model. This involves setting minimum and maximum token budgets for the reasoning process, potentially inducing self-correction (Figure 2).

Our work aims to adapt and evaluate techniques like budget forcing and compare prompting strategies (CoT vs. CoD) in the context of SLMs.

3. Dataset

We evaluate model performance on the following reasoning-intensive benchmark datasets:

1. MATH-500: Mathematical reasoning problems.
2. AIME24: Competition-level Olympiad mathematics problems.
3. GPQA: Graduate-level science questions and answers.

4. Background and Motivation

4.1 What makes a model a Reasoning Model?

What's the difference between an Instruct and Reasoning Model? The difference is just the way these models generate output. Reasoning Model tends to follow a fixed output format i.e. starting with <think> token and then generating the reasoning tokens and then </think> close think tag and then giving the final answer inside <answer> </answer> tag. Here we can think of the Reasoning done as COT Trajectory. Following this idea, the S1 Simple Test Time Compute paper prepared a human curated 1000 Sample Dataset consisting of various domain problems where the COT Trajectory was generated by Deepseek. Now since the output was generated by a reasoning Model, it followed its Reasoning Template. So Fine Tuning an instruct Model on that S1K Dataset will learn the model to follow that reasoning format.

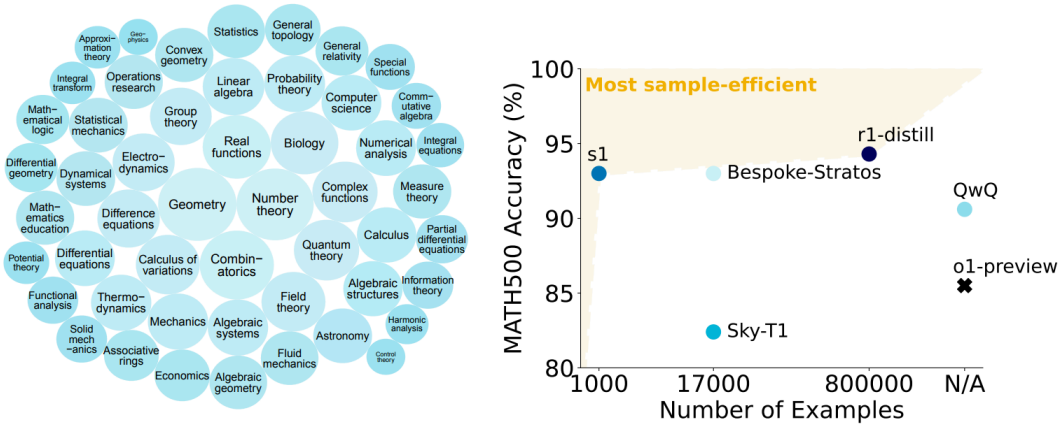


Figure 2. **s1K and s1-32B.** *(left)* **s1K** is a dataset of 1,000 high-quality, diverse, and difficult questions with reasoning traces. *(right)* **s1-32B**, a 32B parameter model finetuned on **s1K** is on the sample-efficiency frontier. See Table 1 for details on other models.

They finetuned Qwen 2.5 32B Instruct Model on S1K Dataset for 5 Epochs and Applied Budget Forcing (will discuss in later section) to scale the number of Thinking Tokens generated.

They were able to boost Score in AIME24 Dataset from 0% to 60%, Math500 from 65% to 95% and GPQA from 40% to 60% by scaling the Thinking tokens 4-20 Times. This shows the potential of Reasoning Models.

4.2 How is Deepseek R1 different from other Reasoning Models?

First thing we agree is that the number of thinking tokens generated by Deepseek R1 is much greater than any other reasoning model present at this time. The primary reason for this is repetitive Self-Reflection in its Thinking Trace. We have evidences of this on Deepseek R1 using the prompt :

Let x,y and z be positive real numbers that satisfy the following system of equations:
\[\log_2\left({x \over yz}\right) = {1 \over 2}\]
\[\log_2\left({y \over xz}\right) = {1 \over 3}\]
\[\log_2\left({z \over xy}\right) = {1 \over 4}\]
Then the value of $\left|\log_2(x^4y^3z^2)\right|$ is $\tfrac{m}{n}$ where m and n are relatively prime positive integers. Find $m+n$.

Simplify that:

$y = y * z^2 * 2^{(1/2 + 1/3)}$

Wait, so $y = y * z^2 * 2^{(5/6)}$. Hmm, divide both sides by y (since y is positive and non-zero), we get:

$1 = z^2 * 2^{(5/6)}$

So the value is 25/8, where m =25 and n=8. Since 25 and 8 are coprime, m + n = 33.

Wait, let me double-check the calculations.

First, $\log_2(x^4y^3z^2) = 4 \log_2 x + 3 \log_2 y + 2 \log_2 z$.

$\log_2 x = -7/24$, so $4*(-7/24) = -28/24 = -7/6 \approx -1.1667$

Deepseek R1 model was able to inherently bring self-reflection ability in its R1 model and called this “AHA Moment”. We will talk about this in more detail under the GRPO Section.

4.3 Budget Forcing (Forcing the Aha Moment)

Budget forcing aims to control the computational budget (amount of "thinking" or reasoning tokens) used by an LM before providing an answer [3]. It works in two main ways:

1. **Maximum Budget (Stopping Early)**: If the model exceeds a specified thinking token limit, an "end-of-thinking" token is forcefully appended, making the model transition to generating its final answer.
2. **Minimum Budget (Thinking Longer / Self-Correction)**: If the model attempts to stop thinking too early (before reaching a desired budget or to encourage more thought), the end-of-thinking token generation is suppressed. Instead, a prompt like "Wait" is appended to the current reasoning trace, encouraging the model to continue reasoning, reflect, double-check, and potentially self-correct errors.

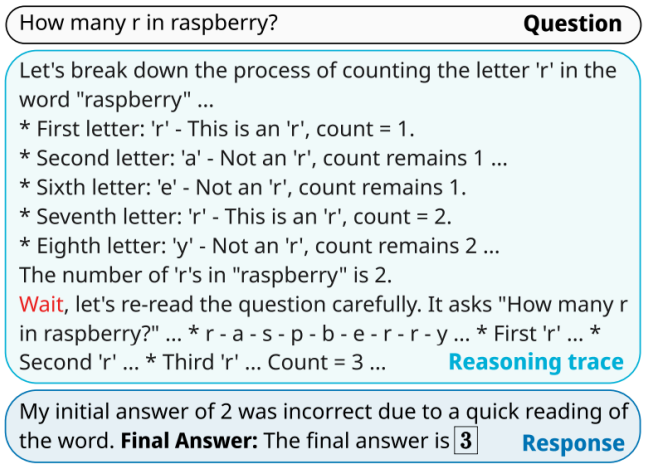


Figure 3. **Budget forcing with s1-32B.** The model tries to stop after “...is 2.”, but we suppress the end-of-thinking token delimiter instead appending “Wait” leading **s1-32B** to self-correct its answer.

Applying budget forcing typically requires the model to follow a specific reasoning format, such as: ‘<think> Reasoning steps... </think> <answer> Final Answer </answer>‘.

4.4 Chain of Thought Vs Chain of Draft

Reasoning models often generate extensive intermediate "thinking" tokens similar to Chain of Thought (CoT) prompting. While helpful for accuracy, this increases latency and computational cost. Chain of Draft (CoD) offers an alternative by prompting the model to produce more concise reasoning steps ("keep a minimum draft...5 words at most" per step).
This aims to balance performance improvements with computational efficiency.

Standard

```
Answer the question directly.  Do not return any
preamble, explanation, or reasoning.
```

Chain-of-Thought

```
Think step by step to answer the following question.
Return the answer at the end of the response after a
separator ####.
```

Chain-of-Draft

```
Think step by step, but only keep a minimum draft for
each thinking step, with 5 words at most. Return the
answer at the end of the response after a separator
####.
```

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	53.3%	1.1	0.6 s
	CoT	95.4%	205.1	4.2 s
	CoD	91.1%	43.9	1.0 s
Claude 3.5 Sonnet	Standard	64.6%	1.1	0.9 s
	CoT	95.8%	190.0	3.1 s
	CoD	91.4%	39.8	1.6 s

Table 1: GSM8K evaluation results.

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	72.6%	5.2	0.6 s
	CoT	90.2%	75.7	1.7 s
	CoD	88.1%	30.2	1.3 s
Claude 3.5 Sonnet	Standard	84.3%	5.2	1.0 s
	CoT	87.0%	172.5	3.2 s
	CoD	89.7%	31.3	1.4 s

Table 2: Date understanding evaluation results.

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	90.0%	1.0	0.4 s
	CoT	95.9%	28.7	0.9 s
	CoD	98.3%	15.0	0.7 s
Claude 3.5 Sonnet	Standard	90.6%	1.0	0.9 s
	CoT	93.2%	189.4	3.6 s
	CoD	97.3%	14.3	1.0 s

Table 3: Sports understanding evaluation results.

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	73.2%	1.0	0.4 s
	CoT	100.0%	52.4	1.4 s
	CoD	100.0%	16.8	0.8 s
Claude 3.5 Sonnet	Standard	85.2%	1.0	1.2 s
	CoT	100.0%	135.3	3.1 s
	CoD	100.0%	18.9	1.6 s

Table 4: Coin flip evaluation results.

Model	Prompt	Accuracy	Token #	Latency
GPT-4o	Standard	56.9%	2.2	0.5 s
	CoT	94.8%	278.4	8.1 s
	CoD	84.4%	76.4	2.6 s
Claude 3.5 Sonnet	Standard	61.9%	5.2	0.9 s
	CoT	90.4%	248.8	3.5 s
	CoD	65.5%	73.7	1.6 s

Table 5: Zero-shot GSM8K evaluation results.

Model	Prompt	Accuracy	Token #
Qwen2.5-1.5B-Instruct	Standard	5.7%	6.6
	CoT	32.5%	141.4
	CoD	24.2%	75.1
Qwen2.5-3B-Instruct	Standard	7.2%	3.4
	CoT	59.1%	236.4
	CoD	43.1%	41.2
Llama3.2-3B-Instruct	Standard	3.9%	16.6
	CoT	70.7%	195.3
	CoD	52.5%	98.1
Zoom-SLM-2.3B	Standard	5.9%	3.8
	CoT	77.7%	129.0
	CoD	50.9%	55.6

Table 6: GSM8K evaluation results on small language models.

4. Methodology

4.1 Supervised Fine-Tuning (SFT)

To enable techniques like budget forcing, the model must reliably adhere to the desired reasoning format (e.g.,‘<think>...</think>‘). One approach is to perform Supervised Fine-Tuning (SFT) on an instruction-tuned model using a dataset containing examples in the target format. The S1 paper [3] utilized a curated 1000-sample dataset (s1K) with CoT reasoning traces generated by a capable model (Deepseek). Fine-tuning an instruct model on s1K teaches it to mimic this reasoning template.

4.2 Group Relative Policy Optimization (GRPO)

GRPO is a technique for aligning language models based on rewards, potentially avoiding the need for explicit CoT examples in the training labels. The model generates multiple candidate solutions, and reward functions provide feedback based on desired criteria (e.g., correctness, format adherence, conciseness). As training progresses, the model learns to generate outputs that maximize the reward.

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>. User: prompt. Assistant:

Table 1 | Template for DeepSeek-R1-Zero. prompt will be replaced with the specific reasoning question during training.

Two ways to use GRPO:

1. Train the model to follow the <think> Reasoning </think> format by first applying SFT, ensuring it learns to respond in the desired thinking format. Then, apply GRPO.
2. Use a system prompt template that specifies the required thinking format for the output.

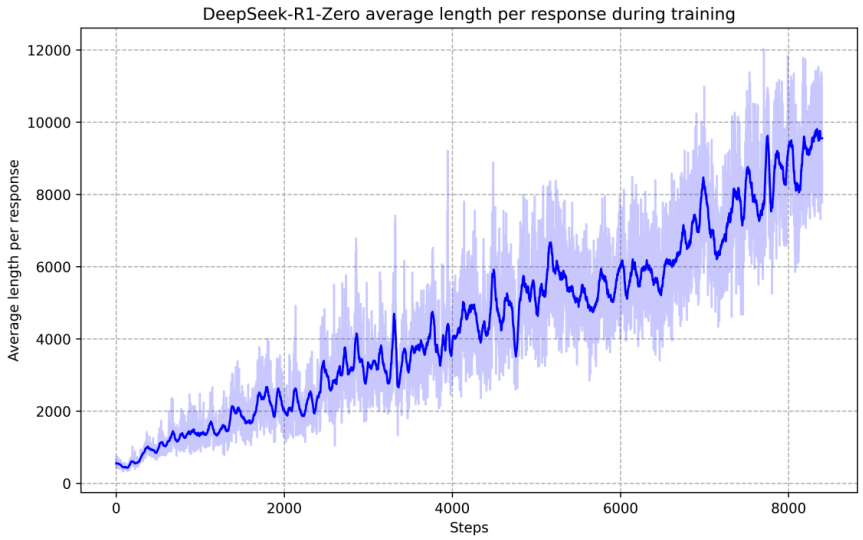
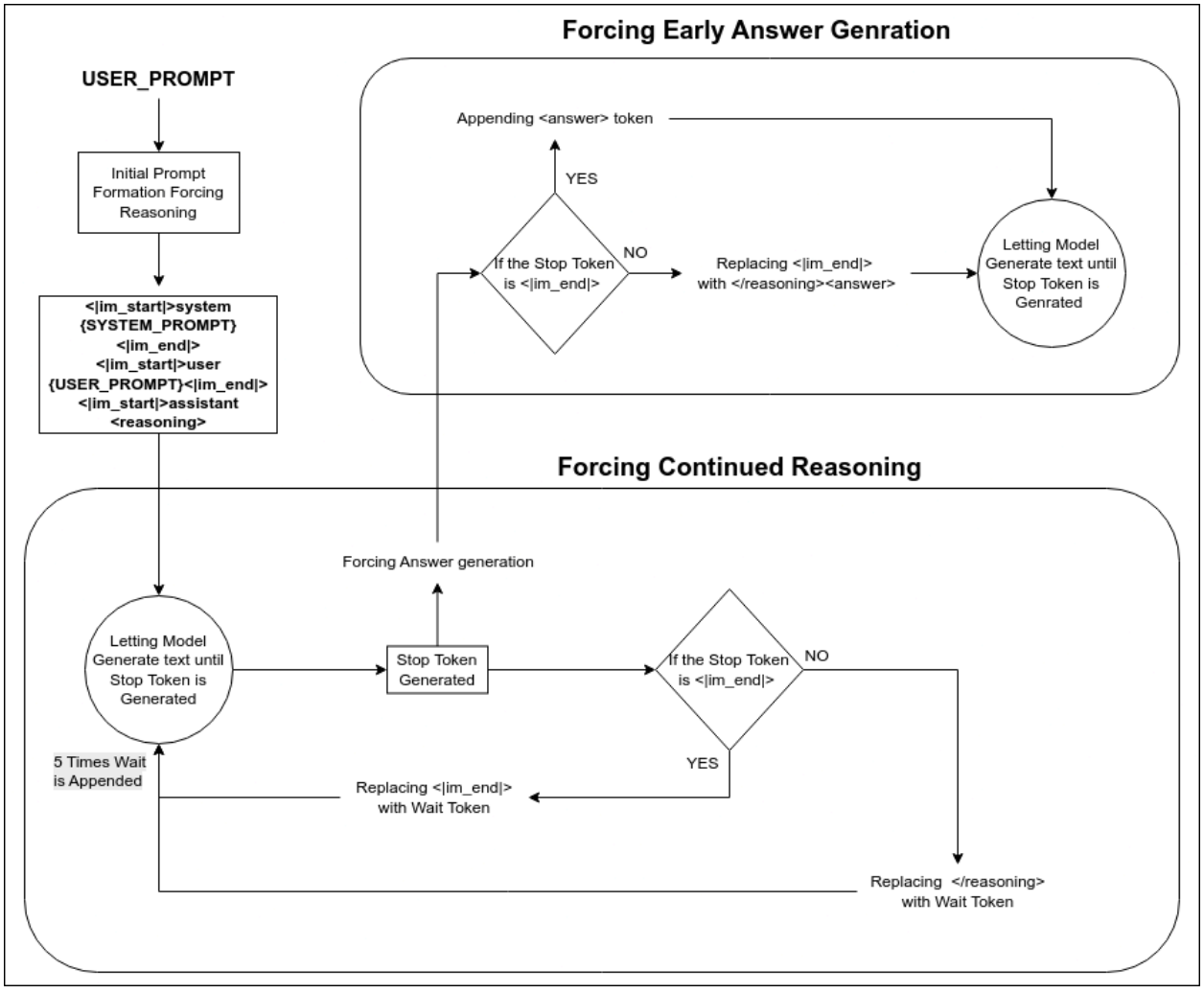


Figure 3 | The average response length of DeepSeek-R1-Zero on the training set during the RL process. DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time.

4.3 Progressive Reasoning Expansion

To better control test-time reasoning, I developed Progressive Reasoning Expansion. Instead of letting model generate whatever its generating till end-of-text (<|im_end|>) is generated and replaced by Wait as per Budget Forcing. In Progressive Reasoning Expansion, i am forcing the first token of the generating to be <reasoning> and setting the stop token as either end-of-text (<|im_end|>) or end-of-reasoning (</reasoning>). If its end-of-text (<|im_end|>), then replacing it by </reasoning><answer> to forcefully generate an answer, also in parallel replace it by Wait, for forcing more reasoning tokens. If it ends by end-of-reasoning (</reasoning>), then appending <answer> for forcing answer generation, and in parallel replacing it by Wait for forcing more reasoning tokens. This helps in forcing any model to follow the reasoning format 100% of the time.



5. Experimentation and Results

Table 2: Response Token Counts Comparison[a,b]

Category	Model	Setting	AIME24	GPQA	Math 500
Prompting on Instruct Models					
	Qwen 2.5-1.5B-instruct	Standard	762	405	852
		Zero Shot	830	807	753
		CoT	1592	775	801
		CoD	444	813	384
	DeepSeek-R1-Distill-Qwen-1.5B	R1	8308	7357	6561
Fine-tuned Models					
	Llama-3.2-1B-instruct[b]	GRPO (temp 0.0)	2948	1247	1649
		GRPO (temp 0.8)	1184	567	751
	Qwen 2.5-1.5B-instruct[a]	SFT (2e-4)	7821	6039	4418
		SFT (1e-4)	7685	4623	6130
		SFT (1e-5)	5338	2598	2357
		SFT (5e-6)	2840	1552	1544
		SFT (1e-6)	1311	796	974
Progressive Reasoning Expansion					
	Phi-4-mini-instruct	Zero Shot	687	642	462
		No Wait	784	753	567
		1 Wait	1136	916	696
		2 Wait	1418	1139	905
		3 Wait	1927	1432	1221

[a] SFT: Fine-tuned on S1K dataset with different learning rates.
[b] GRPO: Fine-tuned on GSM8K dataset with different temperature settings in inference.

Table 1: Model Accuracy Comparison with Improved Formatting[a,b]

Category	Model	Setting	AIME24	GPQA	Math 500
Prompting on Instruct Models					
	Qwen 2.5-1.5B-instruct	Standard	3.33	18.97	25.00
		Zero Shot	0.00	19.20	35.80
		CoT	6.67	17.86	38.00
		CoD	0.00	20.76	26.60
	DeepSeek-R1-Distill-Qwen-1.5B	Zero Shot	3.33	5.36	45.20
Fine-tuned Models					
	Llama-3.2-1B-instruct[b]	GRPO (temp 0.0)	0.00	19.87	33.80
		GRPO (temp 0.8)	0.00	17.86	29.80
	Qwen 2.5-1.5B-instruct[a]	SFT (2e-4)	3.33	12.05	35.20
		SFT (1e-4)	0.00	13.39	29.00
		SFT (1e-5)	0.00	16.74	34.00
		SFT (5e-6)	0.00	20.54	38.00
		SFT (1e-6)	3.33	17.19	35.40
Progressive Reasoning Expansion					
	Phi-4-mini-instruct	Zero Shot	3.33	30.13	51.40
		No Wait	6.66	24.10	53.80
		1 Wait	6.66	23.88	58.00
		2 Wait	6.66	26.33	60.60
		3 Wait	10.00	27.00	64.00

[a] SFT: Fine-tuned on S1K dataset with different learning rates
[b] GRPO: Fine-tuned on GSM8K dataset with different temperature settings inferences
Note: All accuracy values are percentages. Shading indicates major category groups.

5.1 Evaluating Existing Reasoning Models

Since for Applying Budget Forcing, the model should follow the Reasoning Format i.e. something like <think> Reasoning </think> <answer> Answer </answer>. Hence we first did Zero Shot Experiment with **Deepseek-R1-Distill-Qwen-1.5b** on the Benchmark Dataset:

Based on the model responses for thinking model on these benchmark dataset, we set the max_new_tokens as 8192 and did Zero Shot Experiment and analysed the number of responses that went above the limit are :

- **AIME24**: All 30 samples (100%) exceeded the token limit.
- **MATH500**: 361 out of 500 samples (72.2%) exceeded the token limit.
- **GPQA**: 379 out of 500 samples (75.8%) exceeded the token limit.

From here we concluded that it is not possible to apply Budget Forcing on **deepseek-r1-distill-qwen-1.5b.** Hence we need a reasoning model that does reasoning and inherently generates less tokens.

5.2 Baseline Prompting Techniques

Now for this we choose Qwen-2.5-1.5B-instruct model, we proceeded with our baseline experiments that are different prompting Techniques :

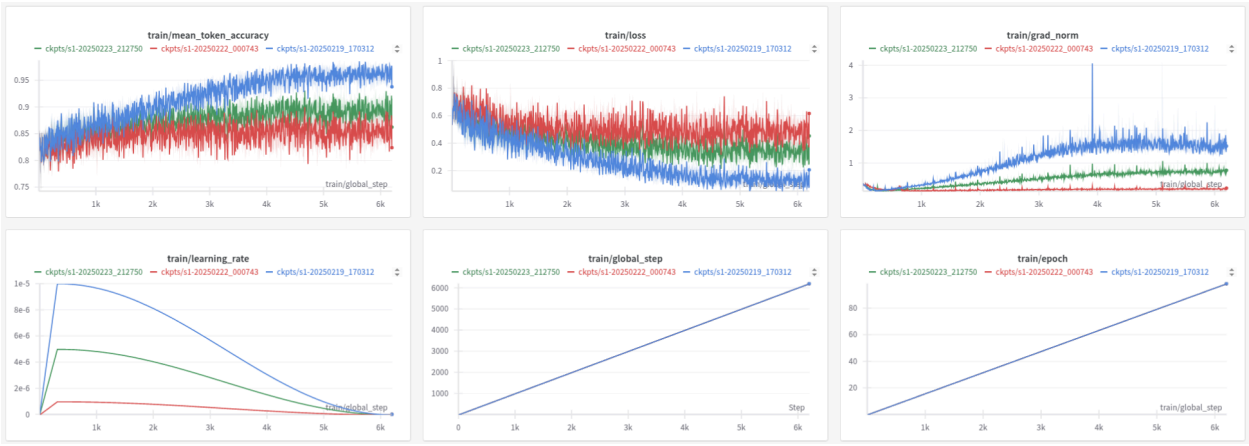
1. Standard
2. Zero Shot
3. Chain of Thought
4. Chain of Draft

The results are in the tables above (Section 5). We saw something strange in this experiment. In the Standard prompting, despite specifically mentioning to directly give the answer and don't produce any other token, it didn't follow it. Although the average Response Token Count is less for Standart, not way higher than what ideally it should be. The best performing promptings were something either COT or COD.

5.3 Making our own Reasoning Model

5.3.1 Fine Tuning Qwen 2.5 1.5B Instruct with SFT on S1K Dataset

I used identical configuration from S1 Paper, like 5 Learning Rates (LRs) Ablation since not much resource and study was done which LRs are usually good for LLMs fine tuning. The LRs that I used were 2e-4, 1e-4, 1e-5, 5e-6 and 1e-6. We observed the below



Here we observed that for higher learning rate, there is a higher train gradient indicating noisy training, but also train loss was less for those running. Hence, here is some quantitative analysis on models SFT trained on the 5 different LRs:

	AIME24			MATH500			GPQA		
Model (LR)	Loops	Tokens	Format	Loops	Tokens	Format	Loops	Tokens	Format
SFT_2e4	24	7821	8	228	4418	220	133	6039	367
SFT_1e4	23	7685	7	253	6130	195	143	4623	357
SFT_1e5	14	5338	0	78	2357	0	76	2598	0
SFT_5e6	6	2840	0	51	1544	0	34	1552	0
SFT_1e6	1	1311	0	22	974	0	4	796	0

Note: 'Loops' refers to the Number of Responses that went into infinite repeating Loop. 'Tokens' refers to the Model Average Response Token Count. 'Format' refers to the Number of Model Responses following the Thinking Output Format.

Findings : We suspect that since the S1K Dataset has a combination of difficult samples whose reasoning traces are around 1000-2000 tokens, maybe SLMs are not effectively able to learn the reasoning. Now what we needed was a Reasoning Model that does reasoning with a reasonable number of tokens so that we can apply Budget Forcing on it. But since the S1K Dataset has a Reasoning Trajectory of slightly longer number of tokens, we decided to use GRPO on a simpler Reasoning Dataset so that model learns to do Reasoning with less tokens. For that we choose the GSM8K Dataset.

5.3.2 Fine Tuning Qwen 2.5 1.5B Instruct with GRPO on GSM8K

We used the same System Prompt that Deepseek used to train its Deepseek R1 Model. But we weren't able to observe the Aha Moment. Basically GRPO wasn't getting any signals (rewards) from the custom reward functions we made. When i observed the Model Responses, i observed it wasn't following the instruction given in the System prompt and always giving output without the Reasoning Format. Then i did some inference on Qwen 2.5 1.5B Instruct with the System prompt saying Always give response by saying i dont know. Surprisingly it didn't even follow this system prompt. I did some digging on the internet and found other people faced similar issues with this particular model.

yuqaf1989 opened on Sep 19, 2024

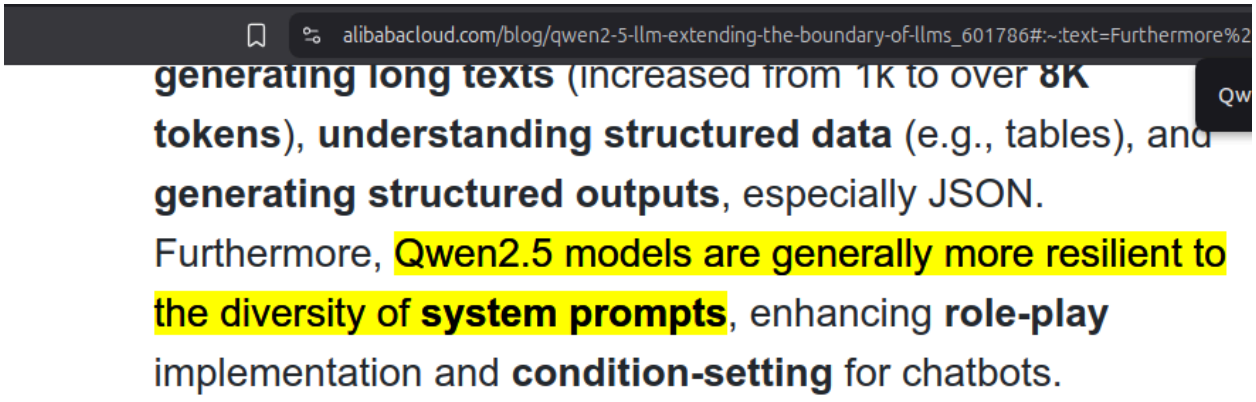
What is the issue?

model: qwen2.5:14b

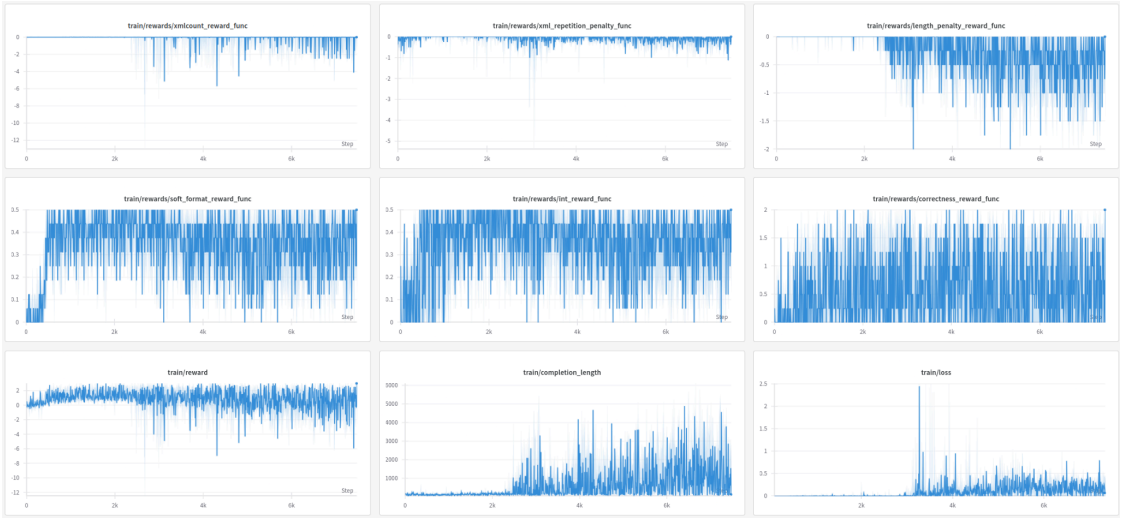
qwen2.5 will ignore system prompt, always return `his name is Qwen.`

test code:

```
from llama_index.llms.ollama import Ollama
from llama_index.core.llms import ChatMessage

sys = ChatMessage(role="system", content="your name is tom.")
c = ChatMessage(role="user", content="who are you")
llm = Ollama(model="qwen2.5:14b", request_timeout=120.0)
llm.chat([sys, c]).message.content

### returns
# I am Qwen, a large language model created by Alibaba Cloud. I am here to help with generating text, answering q

llm = Ollama(model="qwen2:7b", request_timeout=120.0)
llm.chat([sys, c]).message.content
### returns
#I am Tom, an AI assistant designed to provide information and assistance across a wide range of topics, answer q
```



5.3 Fine Tuning Llama3.2-1B-Instruct with GRPO on GSM8K Dataset



Findings : We used the following reward functions :

1. Penalizing Rewards
    a. XMLCount
    b. XML Repetition Penalty
    c. Length Penalty
2. Positive Rewards
    a. Soft Format Reward
    b. Int Reward
    c. Correctness Reward

We observed upto 2500 Steps, the training was stable, but then something happened the the model suddenly started giving very long responses. Despite the penalization given for giving long answers, it didn't help and when I saw the long responses it was giving, it was in many cases gibberish. Hence we took the model from the 2000 Steps Checkpoint and did some analysis.

Our Finding from here is the best 1B to 2B parameters model that are available as of April 2025, are not capable of good Reasoning. They Struggle in following something specific, and are not consistent.

Next we experimented using a little bigger model and tried Phi 4 Instruct 3.8B. We used the System Prompt that Deepseek used, and we found some interesting results in Zero Shot Setting. It was pretty good in following the Reasoning format via the system prompt. We tried fine tuning it, but with limited Compute we weren't able to do it. Hence we applied Budget Forcing on Phi 4 Directly since it was following the System Prompt really well.

5.4 Using Progressive Reasoning Expansion on Phi-4-mini-instruct

Recognizing the format adherence challenges with 1.5B models, we tested Phi-4-mini-instruct (3.8B), which showed better zero-shot compliance with the reasoning template. Applying

Progressive Reasoning Expansion directly, forcing initial reasoning tokens and additional "Wait" steps to extend thinking time, yielded substantial accuracy improvements across benchmarks, notably a +12.6% absolute gain on Math 500 with three "Wait" interventions, correlating with increased token counts. This confirms that for sufficiently capable and instruction-following SLMs like Phi-4, forcing longer test-time reasoning can effectively enhance performance and potentially induce self-correction, validating the approach when base model stability permits.

6. Discussion

Our exploration of test-time optimization for SLMs revealed several key challenges and insights. While techniques like budget forcing show theoretical promise, inspired by observations of self-correction ("Aha Moments") and results from larger models, their practical application to SLMs is significantly hindered by the limitations of current models, particularly those in the 1B-4B parameter range investigated here.

A major recurring hurdle was achieving consistent and reliable adherence to the structured reasoning format (e.g., `<think>...</think>`) required by budget forcing mechanisms. Our SFT attempts with Qwen-1.5B highlighted a difficult trade-off: lower learning rates failed to instill the format, while higher learning rates led to severe instability (loops, excessive verbosity). This suggests that for some SLMs, forcing adherence to a complex generation template via SFT can be disruptive.

GRPO presented its own set of difficulties. Model-specific issues, such as Qwen's tendency to ignore system prompts, made prompt-based format guidance ineffective. Even when the model (Llama-3.2-1B) initially responded to training, the alignment proved unstable, leading to degenerate outputs. This suggests that crafting effective and stable reward functions for complex, multi-objective tasks (correctness, format, conciseness) is particularly challenging for SLMs with limited capacity.

The relative success with Phi-4-Instruct-3.8B in following the prompted format zero-shot, allowing for the direct application of Progressive Reasoning Expansion, was encouraging. It implies that the inherent instruction-following capabilities and stability of the base SLM are critical prerequisites for these test-time techniques. For models that possess these qualities, direct application of budget forcing might be a more pragmatic approach than complex alignment procedures like SFT or GRPO, especially under typical resource constraints for SLM usage.

The concept of the "Aha Moment" remains a compelling motivation for encouraging longer reasoning times via Minimum Budget Forcing. Our Progressive Reasoning Expansion mechanism is a direct attempt to operationalize this. However, its effectiveness is clearly contingent not only on format adherence but also on the model's baseline reasoning ability. Simply forcing more tokens does not guarantee better reasoning; the quality of the extended thought process is paramount.

7. Conclusion and Takeaways

Optimizing test-time compute allocation holds significant potential for improving the reasoning abilities of SLMs, bridging the gap between their knowledge and their ability to apply it effectively. Our investigation into budget forcing, SFT, GRPO, and prompting strategies like CoD highlights both the opportunities and the substantial challenges involved in realizing this potential with current SLM technology.

Key Takeaways:

1. **Format Adherence is Key:** Current SLMs (particularly 1B-2B models tested) struggle significantly with the consistency and stability required for advanced test-time optimization techniques like budget forcing, often failing to adhere reliably to specific output formats.
2. **Model Dependencies Matter:** Base model characteristics are crucial. Issues like Qwen-2.5 models potentially ignoring system prompts hinder prompt-based control and alignment methods like GRPO that rely on them.
3. **SFT is Delicate:** While SFT can potentially instill reasoning formats, it requires careful tuning for SLMs, as stability issues (loops, verbosity) can easily arise, sometimes correlated with successful format adoption.
4. **GRPO is Challenging for SLMs:** GRPO is complex to implement effectively, sensitive to reward function design, and prone to training instability or reward hacking, particularly with lower-capacity models.
5. **Better Instruction Following Helps:** SLMs with stronger inherent instruction following capabilities (like Phi-4 observed here) might be more amenable to direct application of budget forcing techniques without extensive, potentially destabilizing, fine-tuning.
6. **Budget Forcing Potential:** Budget forcing, especially methods encouraging longer reasoning (Minimum Budget / Progressive Reasoning Expansion), conceptually aligns with fostering self-correction ("Aha Moments"). However, its success critically depends on the model's baseline reasoning quality and its ability to maintain coherence during forced continuation.

8. Future Work

Future work should prioritize the development of SLMs with improved inherent instruction following, reasoning capabilities, and training stability. These advancements would create a stronger foundation for applying test-time optimization techniques. Additionally, exploring simpler, potentially prompt-based, or more robust methods for controlling computational budget and inducing reflection in SLMs remains an important and promising research direction.