Less is More: Compressed Reasoning with Large Language Models via Structured Prompting

Anonymous authors
Paper under double-blind review

Abstract

Chain-of-thought (CoT) prompting significantly improves LLM performance on reasoning and mathematical tasks by enabling transformers to perform complex computations across multiple inference steps. Modern pre- and post-training pipelines have amplified this: LLMs are now post-trained with explicit CoT supervision during SFT and further reinforced via RL to produce longer, more structured reasoning chains, achieving state-of-the-art results on benchmarks such as MATH-500, GSM8K, and ARC-Challenge, but at the cost of significantly increased token generation. In this paper, we theoretically analyse the reason CoT improves the ability of LLMs to perform reasoning tasks, and identify two properties - step-complexity and step-expressiveness - that any compressed reasoning format must preserve to maintain CoT-level performance. We show that existing concise prompting strategies fail because they violate one or both of these properties. As an instantiation of our analysis, we introduce TeleMathLang (TML), a minimal structured syntax for reasoning that enables LLMs to generate complete reasoning chains while reducing response length by 25–69% across MATH-500, GSM8K, and ARC-Challenge, without requiring further post-training which could be expensive and time-consuming. TML works as a prompting strategy with any instruction-tuned model, and we further show via conditional generation that models already trained via SFT and RL to produce long reasoning chains can be efficiently fine-tuned to generate TML output without accuracy loss. TML outperforms existing concise prompting baselines in accuracy across all three benchmarks.

1 Introduction

Chain-of-thought (CoT) prompting (Wei et al., 2023) has become one of the most widely-used techniques for improving LLM performance on reasoning tasks. It allows an LLM to generate intermediate steps before producing an answer, leading to substantially higher accuracy on mathematical and logical problems compared to responses without COT. Building on this, modern post-training pipelines have amplified CoT further. COT is further reinforced in modern models in multiple post-training stages. For example, Qwen models go through supervised fine-tuning on examples including chain-of-thought mathematical data from Qwen2.5-Math (Section 4.1, Qwen et al. 2025), and online reinforcement learning is done via Group Relative Policy Optimization (GRPO; Section 4.3, Qwen et al. 2025), which further incentivises structured, step-by-step reasoning chains. DeepSeek-R1 (DeepSeek-AI et al., 2025) and OpenAI o1 (OpenAI et al., 2024) follow similar paradigms. The result is a generation of models that reason more reliably, but produce substantially longer outputs, with CoT chains often running to thousands of tokens even for problems that require only a handful of reasoning steps.

However, the added verbosity in CoT-based responses lead to lengthy intermediate explanations (Aggarwal & Welleck, 2025; Fatemi et al., 2025), that inflate inference cost and latency. Attempts to address this directly by specifying output token budgets or fixing the number of reasoning steps consistently degrade performance (Jin et al., 2024): the model produces shorter chains but answers fewer questions correctly. The relationship between chain length and accuracy illustrated in Figure 1 makes this trade-off concrete.

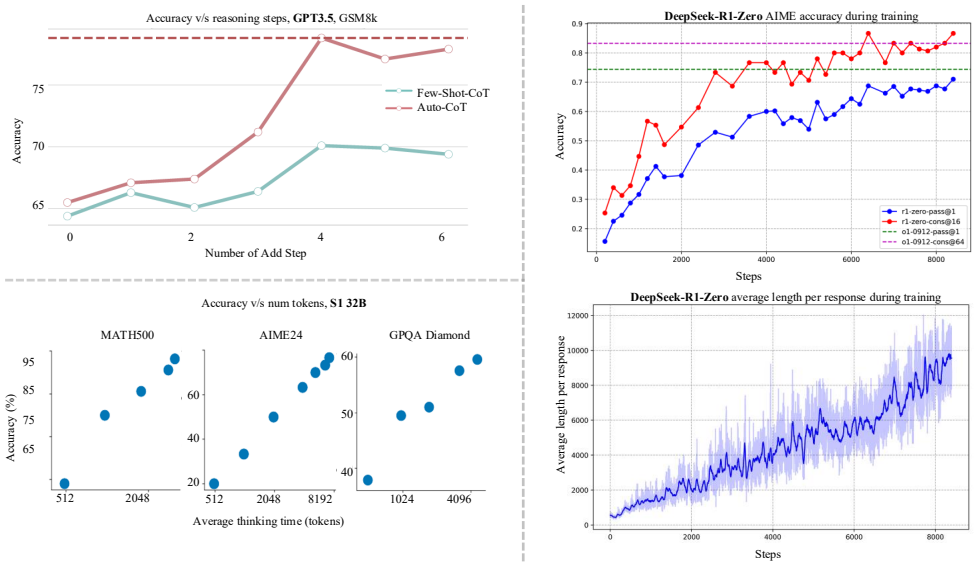

Figure 1: Correlation between large response size and accuracy in recent works. Top left from Jin et al. (2024) bottom left from Muennighoff et al. (2025), right from DeepSeek-AI et al. (2025)

One line of prior work has explored reinforcement learning techniques to shorten reasoning chains while preserving accuracy. Specialised modifications to GRPO have been proposed with reward functions that penalise overly long solutions (Fatemi et al., 2025; Aggarwal & Welleck, 2025; Liu et al., 2025; Hou et al., 2025). In practice these methods can compress chains-of-thought, but they are computationally expensive. For instance, GRPO on a 1.5B model with 6 generations per step can require 4 × 48GB GPUs to complete one epoch on 40K samples (Dang & Ngo, 2025). Additionally, these methods can be unstable and brittle across hyperparameter choices (Hochlehnert et al., 2025). RL objectives aimed at length reduction also risk reward hacking (Gao et al., 2022)., where models may learn superficially shorter outputs rather than genuinely more efficient reasoning

A third line of work imposes explicit structure on reasoning chains to improve quality. Hi-CoT (Huang et al., 2026) alternates between high-level instruction planning and step-level execution, improving accuracy on mathematical benchmarks. This is consistent with the step-complexity framework we develop in Section 3: by enforcing serial decomposition, Hi-CoT preserves step count and avoids accuracy degradation. However, Hi-CoT does not reduce token count in standard instruction-tuned models; savings appear only in extended-thinking models that already generate thousands of tokens. This confirms that structure alone is not sufficient to reliably reduce token count while preserving accuracy.

In this paper, we analyse why CoT improves reasoning ability and identify two properties, step-complexity and step-expressiveness, that a compressed reasoning format must preserve to maintain CoT-level performance (Section 3). This motivates TeleMathLang (TML), a minimal structured syntax that preserves both properties while reducing per-step verbosity. TML works as a prompting strategy for any instruction-tuned model with no fine-tuning required, achieving 25–69% token reductions at CoT-level accuracy across four LLMs and three benchmarks. For models post-trained to generate long reasoning chains (Qwen 2.5 Qwen et al. 2025), prompting alone is insufficient; we show that conditional generation (Kang et al., 2024) on a small TML dataset recovers CoT-level accuracy with substantial token savings, without expensive RL.

To summarize, our contributions are the following -

- We theoretically analyse why CoT improves LLM reasoning ability, and identify step-complexity and step-expressiveness as the two properties a compressed format must preserve. We show why existing concise prompting strategies violate these properties and thereby degrade accuracy.

- We introduce TeleMathLang (TML): a minimal structured reasoning syntax that preserves both properties while reducing token count. As a prompting strategy, TML achieves CoT-level accuracy on MATH-500, GSM8K, and ARC-Challenge with 25–69% fewer tokens, consistently outperforming BeConcise, NoProperGrammar, Chain-of-Draft, and Hi-CoT in token efficiency across four instruction-tuned LLMs.

- We demonstrate conditional generation as a resource-efficient alternative to RL for models already trained to produce long reasoning chains: fine-tuning Qwen 2.5 models on a small TML-formatted dataset recovers CoT-level accuracy with substantial token savings, without iterative RL training.

2 Related Work

LLM Reasoning Brown et.al. (Brown et al., 2020) showed that LLMs are able to generalize to unseen tasks if few illustrative examples were included in the prompt. Chain-of-Thought (CoT) prompting (Wei et al., 2023) significantly improved LLM performance in tasks that required reasoning, through the usage of few-shot examples that showed multi-step reasoning. By breaking down complex problems into smaller, manageable steps, CoT allows LLMs to generate chains of reason that lead to the correct answer (discussed in more detail in section 3). Kojima et al. (Kojima et al., 2023) showed that models can be encouraged to generate chains of thoughts even without few-shot examples. More sophisticated structured reasoning methods have been proposed more recently (Yao et al., 2023a; Chen et al., 2023; Xu et al., 2025; Yao et al., 2023b), which attempt to increase accuracy by maintaining multiple reasoning chains, using computational tools, or multi-step reason-observe-act chains. However, CoT remains widely used. More recently, thinking models have been developed that are explicitly designed to think longer during inference (DeepSeek-AI et al., 2025; Qwen et al., 2025; Shao et al., 2024; OpenAI et al., 2024). These models are trained on high-quality supervised fine-tuning (SFT) data with responses in CoT and program-of-thought (PoT) format (Chen et al., 2023). SFT checkpoints are then further incentivized to generate longer CoT through reinforcement learning (RL). DeepSeek R1 (DeepSeek-AI et al., 2025) reported that during extensive reinforcement learning, the model learned to correct itself and generate longer chains. However, CoT-based training biases models toward high verbosity: RL objectives can incentivise long chains even when intermediate steps are suboptimal (Liu et al., 2025), creating longer chains that are not always correct (Aggarwal & Welleck, 2025; Fatemi et al., 2025).

Concise Reasoning To address the verbosity in CoT responses, studies have explored concise prompting strategies. Lee et al. (2025) explore the performance of a host of prompting strategies across multiple LLMs. Cheng & Durme (2024) propose dense contemplation tokens which are novel but show significantly poorer accuracy on GSM8K based on reported performances. Xu et al. (2025) propose chain-of-draft, where LLMs are instructed to limit each thinking token to 5 words at most. Kang et al. (2024) proposes finetuning on a small training dataset with short-form CoT solutions (C3oT); Wu et al. (2025) finetune on a mix of reasoning and non-reasoning data. While these approaches are promising, they have limitations. Chain-of-draft shows noticeably poor performance in small LLMs, while C3oT requires finetuning and will not work with proprietary LLMs, making both these approaches less generalizable than ours. We discuss theoretically why concise prompting is suboptimal in Section 3, and show empirically that TML outperforms these strategies on hard reasoning tasks (Section 5).

Several studies have also explored reinforcement learning (RL) approaches aimed at producing more concise reasoning paths. Liu et al. (2025) propose a modification to GRPO by removing normalization term in the objective. Aggarwal & Welleck (2025); Fatemi et al. (2025) propose modifications to GRPO using token constraints, and Hou et al. (2025) uses an iterative approach to reduce token count gradually. While these approaches show promise, they often require substantial computational resources and can be unstable during training. Additionally, the gains made by RL-based methods are brittle, as shown by Hochlehnert et al. (2025). In this paper, we will show that TeleMathLang is able to generate concise reasoning without significant computational overhead consistently across LLMs and datasets.

3 Theoretical Motivation: Why Concise Prompting Fails

Background. The transformer architecture allows computations over tens of thousands of tokens. However, the expressivity of a single computation is bounded by low-complexity circuit classes: a single forward pass is simulable by TC^0 circuits (constant depth, threshold gates) under log-precision arithmetic (Merrill & Sabharwal, 2023). CoT overcomes this limitation by using each generated segment as read-again input in the next computation, turning one fixed-depth computation into a sequence of shallow computations(Figure 3) (Li et al., 2024). The key result shown in that paper is that increasing the number of CoT steps increases transformer expressivity: with poly(n) steps, a transformer can solve all of P/poly — the full class of problems solvable by polynomial-size circuits (Li et al., 2024). To reiterate, this increase in expressivity is caused by *step count*, not raw token count, which is why Lee et al. (2025) found that increased token count by adding 100 whitespace tokens etc do not lead to improved accuracy.

3.1 Step-Complexity

Lee et al. (2025) observe sharp accuracy cliffs when chain length falls below problem-specific thresholds on GSM8K and MATH-500. We sharpen this into a *step-level* hypothesis by defining a step precisely:

Definition (Read-after-write step). *Partition the generated output into segments $s_1, s_2, \ldots$. Segment s_t is a* ***read-after-write step*** *iff it computes at least one boolean circuit gate value, and $\exists s_u$, $u > t$, such that s_t has a causal effect on s_u.*

Appending whitespace or filler tokens does not increase step count, since these compute no additional gate value. Note that this definition only holds for binary strings. For natural language problems, our definition holds if we replace the target answer with its binary representation. In that case, each CoT step can be thought of as performing one operation towards that target representation, which is the strategy used by Li et al. (2024) to prove that with poly(n) steps, a transformer can solve all of P/poly (theorem 3.3 in the paper).

The **step-complexity hypothesis** states: for an input x, there exists a minimum number of read-after-write steps $D^*(x)$ such that a fixed-depth transformer cannot solve x with fewer. Reducing step count degrades accuracy.

We corroborate this empirically on MATH-500, which is annotated by difficulty level. We took the available solutions, and generated solutions using 3 different LLMs (Claude 3.5v1, 3.7 and Nova Pro) under 3 different conditions (with instructions to reason step-by-step, to be concise, and finally to be very concise). We generated step labels using Claude 4.5 Sonnet, a neutral LLM, where each step was defined as an indecomposible operation required in a subsequent operation. Figure 2 shows the results we saw: step count increases monotonically with difficulty level, and forcing concise or very-concise outputs reduces both step count and accuracy across all difficulty levels.

3.2 Step-Expressiveness

By definition, each step **must compute at least one gate value given the input and previously generated segments**. We call this property **step-expressiveness**.

3.3 Why Concise Prompting Fails

General concise instructions such as "Be Concise" or "avoid proper grammar" reduce step count without providing a mechanism to preserve serial decomposition, violating step-complexity and degrading accuracy on hard problems. **Chain-of-Draft** (Xu et al., 2025) enforces a five-word-per-step limit, capping the expressiveness of each intermediate state and violating step-expressiveness on hard mathematical problems; this is acknowledged in the original paper and corroborated by our findings in Section 5.1.3.

Hi-CoT (Huang et al., 2026) takes a different approach: it enforces serial decomposition by alternating between high-level planning and step-level execution, thereby preserving step count and avoiding accuracy degradation. However, Hi-CoT does not reduce per-step verbosity, and token savings appear only in

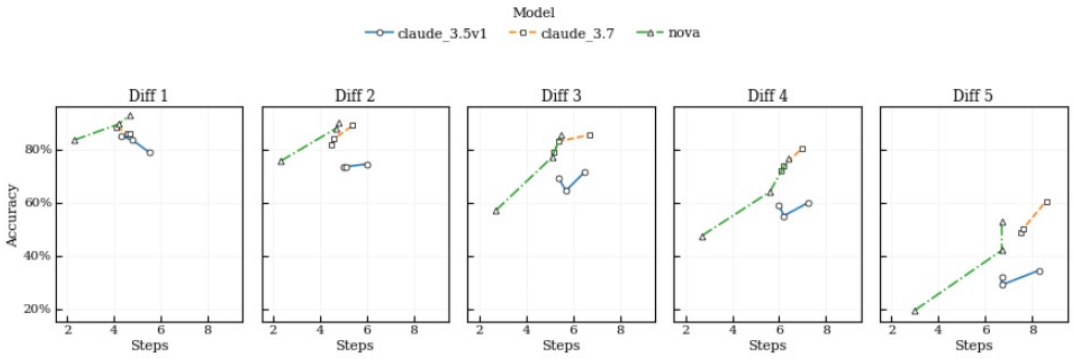

Figure 2: Accuracy vs. number of reasoning steps across MATH-500 difficulty levels (1–5) for three models (Claude 3.5v1, Claude 3.7, Nova Pro). Step count increases monotonically with difficulty, and reducing steps (CoT → Concise → VeryConcise) consistently reduces accuracy — confirming that step count, not token count, drives performance.

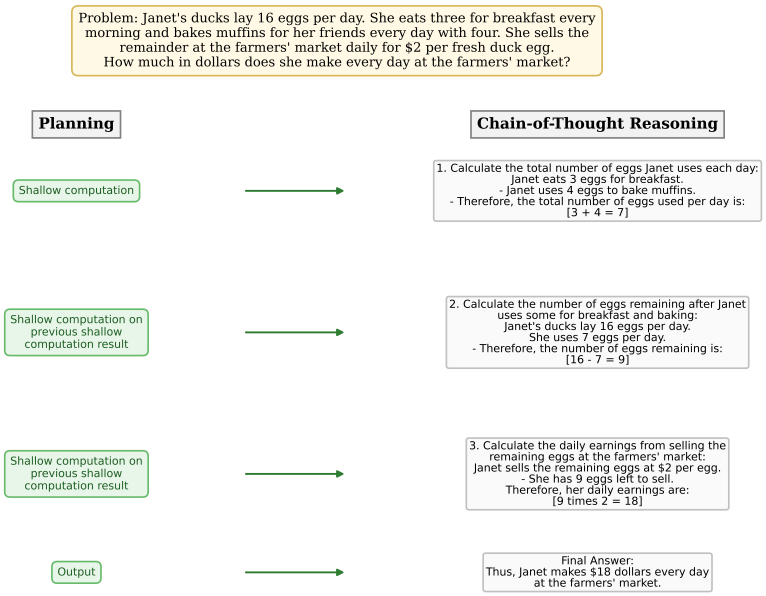

Figure 3: Step-by-step reasoning and shallow computation

extended-thinking models that already generate thousands of tokens. TML addresses the gap present in concise prompting, preserving both step-complexity and step-expressiveness while compressing per-step token count, achieving efficiency in any instruction-tuned model without requiring a specialised extended-thinking mode.

4 How can we generate token-efficient reasoning chains?

The step-complexity and step-expressiveness requirements from Section 3 constrain what any valid compressed reasoning format must preserve: adaptive serial decomposition into read-after-write steps, each

Core Reasoning Keywords		Extended Reasoning Keywords		Syntax Rules
Given:	facts and setup	**Approach:**	strategy selection	Begin each line with one of the keywords
Let:	define variables	**Case:**	case analysis	Use LaTeX for mathematical notation
Then:	next logical step	**Insight:**	key observations	
So:	intermediate result	**Assume:**	for assumptions	(e.g., 2×x, (a + b), x^2)
Check:	verify logic	**Lemma:**	supporting claims	Evaluate expressions when possible
Therefore:	conclusion	**Contradiction:**	proof method	(e.g., "Let x = 3×2 = 6"),
Verify:	double check	**Induction:**	inductive steps	showing calculation
Answer:	final result	**Generalize:**	extending patterns	

Table 1: TeleMathLang Syntax

expressive enough to encode a usable intermediate state. We propose **TeleMathLang (TML)** as one instantiation of these requirements.[1]

As motivation, consider Wei et al. (2023); Ye et al. (2025); Muennighoff et al. (2025); DeepSeek-AI et al. (2025), which show the association between the existence of logical connectors (therefore, since) and encouragement for serial computation ("Let's think step by step"), with increased reasoning steps (and accuracy). In our experiments, we focused on using vocabulary from formal mathematical proof writing where breaking problems into steps is a natural way of solving. Additionally, modern LLMs like Deepseek, DeepSeek-AI et al. (2025), Qwen Qwen et al. (2025) etc are pre-trained on large corpuses of mathematical problems. What distinguishes TML from prior structured prompts is step expressiveness: Chain of Draft (Xu et al., 2025) enforces a five-word-per-step limit, conflicting with the step-expressiveness requirement on hard math. TML focuses more on breaking problems up, and does not constrain not how much each step may say.

We define TeleMathLang as a syntax with a **fixed set of reasoning keywords that can be used to begin a sentence**, and **rules to ensure reasoning keywords are used** (Table 1). These keywords are chosen to force breaking problems into steps and sequential problem solving, avoiding the limitations of concise prompting instructions. We divide the keywords into core and extended, where core keywords are very commonly used and extended keywords are used in more complex problems. We find that this encourages models to use core keywords more extensively in simpler problems and use extended keywords in more complex problems to break those down into more steps, which we show in our ablations. These set of keywords cover a large range of logic and math problems, and can be extended for particular usecases.

Prompting In few-shot or zero shot setup with foundational models, the prompt provided consists of a description of TeleMathLang as an **ultra-minimal syntax**, the **keywords** and **rules**, few-shot **labelled** examples (or zero-shot, as is the case) and the instruction "Please think step-by-step in TeleMathLang, and return your final answer in boxed{answer}". The few-shot labelled examples are kept fixed across all evaluation benchmarks to assess generalizability of our method. The full prompt template and few-shot examples can be found in appendix C and D. Note that TeleMathLang satisfies both step-expressivity and step-complexity. By emphasising the reasoning keywords, LLMs are encouraged to generate as many reasoning steps as required, while the minimal math-notation-based language is sufficiently expressive, satisfying grammar rules seen in pre-training corpus of most LLMs. Moreover, step-level length is adaptive, and no restrictions are imposed. In Section 5 we show that motivating TeleMathLang as an ultra-minimal syntax consistently reduces token count while preserving accuracy. Figure 4 shows an example of CoT vs. TeleMathLang where the number of reasoning steps is higher in TeleMathLang while total token count is lower.

Finetuning We use TeleMathLang in a few-shot prompting setting as well as a finetuning setting. For finetuning, TeleMathLang responses are prepared for the train dataset using a state-of-the-art LLM, generally Claude Sonnet 4.5 (Anthropic, 2025). To leverage the familiarity of recent LLMs with CoT due finetuning on CoT samples, we carry out conditioned training (Kang et al., 2024). We create a train dataset comprising

[1]Circuit-complexity theory constrains the functional roles a compressed format must preserve (serial decomposition and expressive intermediate steps) but does not uniquely determine vocabulary. TML is one principled instantiation, not the only theoretically valid keyword set.

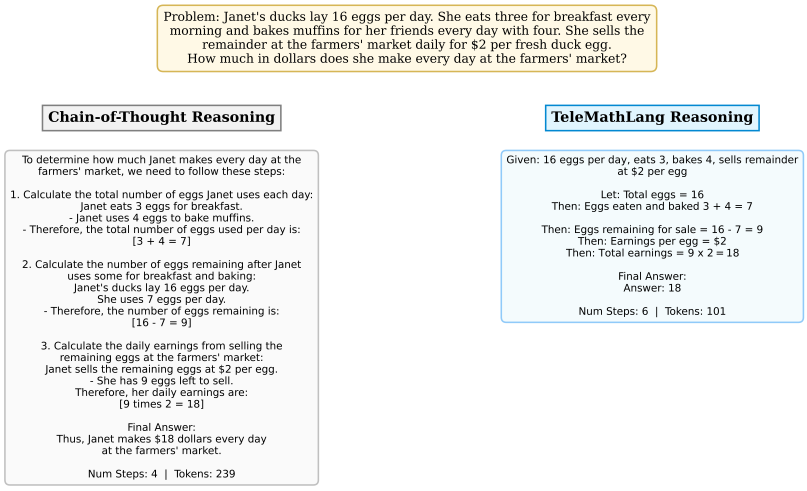

Figure 4: CoT v/s TeleMathLang reasoning chains

of CoT as well as TeleMathLang demonstrations, creating a class-conditioned fine-tuning dataset (with one class for CoT and one for TeleMathLang, distinguished by the associated prompt). The purpose of this is to allow LLMs to learn the correspondence between CoT and TeleMathLang reasoning. During inference, we provide the TeleMathLang prompt to get concise reasoning and response. More descriptions of labelling, training and testing are present in Section 5.

5 Experimental Setup and Results

Model	MATH-500			GSM8K			ARC-Challenge			Average
	CoT	TML	Token↓	CoT	TML	Token↓	CoT	TML	Token↓	Token↓
Sonnet 3.5 v1	54.00%	**58.67**%	61.33%	89.66%	**95.48**%	69.21%	95.09%	94.65%	24.10%	51.55%
Sonnet 4.5	90.8%	90.4%	29.1%	97.8%	97.7%	45.8%	97.6%	97.6%	48.2%	37.9%
Nova Pro	72.87%	69.33%	46.52%	95.07%	93.91%	53.68%	93.87%	93.65%	46.33%	48.84%
Gemma 3 27B	76.50%	**78.50%**	35.88%	93.44%	93.33%	26.40%	92.30%	91.42%	32.46%	31.58%

Table 2: Performance comparison between CoT and TeleMathLang (TML) across datasets averaged across 0, 4 and 8 shot settings, showing accuracy and token reduction (Token↓).TeleMathLang maintains comparable accuracy while significantly reducing token count. Instances where it outperforms CoT are in bold.

5.1 Experimental Setup

Models and Benchmark Datasets We use three external benchmark datasets focusing on math and reasoning: MATH-500 (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021) and the ARC Challenge set (Clark et al., 2018). While GSM8K contains grade school math problems, MATH-500 is more challenging, containing competition math problems. ARC-Challenge contains multiple-choice science questions requiring multi-step reasoning, and is substantially harder than the ARC-Easy split. Combined, these benchmarks show an LLM's ability to reason, calculate and solve sequential problems.

Models To assess the generalizability of our approach across LLMs we run experiments across 0-shot, 4-shot, and 8-shot settings on Claude 3.5 Sonnet v1 (Anthropic, 2024), Claude Sonnet 4.5 (Anthropic, 2025), Nova Pro (Intelligence, 2024) and Gemma 3 27B (Team et al., 2025). We report averages across all three settings rather than the best-performing setting to demonstrate that TML's token reduction and accuracy gains

Condition	MATH-500		GSM8K	
	Acc.	Token↓	Acc.	Token↓
CoT	90.6%	0.0%	98.0%	0.0%
TML (full, v4a)	**91.0%**	30.4%	97.7%	41.7%
TML-Core (8 core keywords)	88.8%	25.4%	97.7%	33.5%
Random-Keywords	88.0%	17.9%	98.0%	16.0%
Generic-Structured (Step N:)	87.4%	34.4%	97.7%	35.6%
Hi-CoT (Huang et al., 2026)	90.0%	18.5%	**98.3%**	15.4%
BeConcise	90.6%	8.2%	**98.3%**	23.7%
NoProperGrammar	90.6%	-4.4%	97.0%	18.1%
Chain-of-Draft	64.2%	40.0%	97.0%	61.3%

Table 3: Ablation study comparing TML against keyword variants and concise prompting baselines on MATH-500 and GSM8K using Claude Sonnet 4.5, 4-shot. Token↓ shows percentage reduction relative to CoT. The middle block isolates the contribution of TML's specific keyword semantics: TML-Core uses only the 8 core keywords; Random-Keywords replaces TML vocabulary with arbitrary structured markers; Generic-Structured uses "Step N:" labels; Hi-CoT imposes hierarchical instruction–execution structure (Huang et al., 2026).

are consistent across prompting regimes. Per-shot results are provided in Appendix E (Table 8). Few-shot examples are drawn from the open-deepscaler dataset (Dang & Ngo, 2025), which contains challenging math problems not present in any of the three evaluation datasets.

Inference Settings All API models are queried at temperature 0.7 (Claude 3.5 Sonnet v1, Claude Sonnet 4.5, Nova Pro), following provider-recommended settings for reasoning tasks. Gemma 3 27B is served via vLLM at temperature 0.2 per the Google Cloud deployment guide (Team et al., 2025). Qwen 2.5 uses Qwen-recommended settings (Qwen, 2024). Answers are extracted by parsing `\boxed{}` in model outputs, following the MATH-500 evaluation convention.

Prompt token overhead Table 4 reports input token counts per configuration. At 0-shot the TML keyword table adds approximately 85 tokens over CoT; at 4- and 8-shot, TML few-shot examples are themselves concise, so the TML prompt is *smaller* than the CoT prompt (555 vs. 757 at 4-shot; 1,153 vs. 1,432 at 8-shot). Net total-token savings (input + output combined) therefore exceed the output-only Token↓ figures reported in Table 2.

Configuration	Input tokens
CoT 0-shot	37
Hi-CoT 0-shot	86
TML 0-shot	122
Hi-CoT 4-shot	515
TML 4-shot	555
CoT 4-shot	757
TML 8-shot	1,153
CoT 8-shot	1,432

Table 4: Input (prompt) token counts per prompting strategy and shot setting. At 4- and 8-shot, TML's concise few-shot examples yield a *smaller* prompt than CoT, so net total-token savings exceed the output-only Token↓ figures.

Finetuning The Claude, Nova and Gemma models are instruction-tuned to be helpful and general purpose. In contrast, models such as the Qwen 2.5 instruct series are post-trained on large corpuses of data with responses in long-CoT format(Qwen et al., 2025). This is done to explicitly increase output length, and the models perform best when prompts match post-training data, and with increased test-time computation

Dataset	CoT		TeleMathLang		BeConcise		NoProperGrammar		Chain-of-Draft	
	Acc.	Token↓	Acc.	Token↓	Acc.	Token↓	Acc.	Token↓	Acc.	Token↓
ARC-Challenge	55.85%	0.00%	44.30%	26.71%	28.42%	26.83%	17.05%	27.95%	6.68%	67.54%
GSM8K	72.23%	0.00%	67.12%	-4.71%	69.21%	24.24%	69.14%	28.94%	51.18%	40.08%

Table 5: Performance comparison of concise promp-based strategies using Qwen 2.5 1.5B. Each cell shows accuracy and token reduction relative to CoT.

Model Size	Dataset	CoT		CoT (FT)		Summ. CoT (FT)		TML (FT)		Cond. Gen.	
		Acc.	Token↓	Acc.	Token↓	Acc.	Token↓	Acc.	Token↓	Acc.	Toke
0.5B	ARC-Challenge	36.12%	0%	41.13%	6.72%	40.80%	49.89%	41.13%	**56.47%**	43.81%	38.17
0.5B	GSM8K	48.14%	0%	47.11%	8.71%	31.15%	59.59%	39.19%	52.14%	46.24%	47.24
1.5B	ARC-Challenge	55.85%	0%	72.24%	8.18%	65.55%	51.96%	70.23%	40.06%	74.92%	38.98
1.5B	GSM8K	71.86%	0%	71.86%	2.07%	36.92%	61.97%	69.85%	52.82%	71.36%	46.29
3B	ARC-Challenge	80.94%	0%	81.27%	0.23%	79.26%	45.77%	76.25%	40.02%	79.93%	35.62
3B	GSM8K	83.16%	0%	81.91%	3.66%	63.68%	57.53%	71.36%	56.61%	80.40%	38.55
7B	ARC-Challenge	87.96%	0%	87.29%	17.04%	85.61%	49.46%	86.62%	46.97%	88.29%	47.05
7B	GSM8K	91.46%	0%	90.45%	-0.82%	58.60%	56.34%	87.43%	56.70%	90.24%	51.43

Table 6: Qwen 2.5 models on ARC-Challenge and GSM8K with various inference strategies. **Token↓** shows the percentage reduction in generated tokens relative to non-fine-tuned CoT. Summ. CoT refers to summarized CoT and Cond. Gen. refers to conditional generation

(Muennighoff et al., 2025). We show that prompting-based methods show poor performance here, and that this can be overcome with finetuning. For Qwen 2.5, we finetune the 1.5B instruction-tuned model in a conditional generation framework as discussed in section 3 and compare against CoT instructions as well as CoT-based finetuning. The implementation details can be found in appendix G.

5.1.1 Performance Analysis

Overall performance results are shown in Table 2. The gains for some models in particular are noticeably large - Claude 3.5 v1 and Nova Pro achieve token count reductions of ∼50%+ over CoT while preserving accuracy. Accuracy is preserved in other models as well, with token count reductions of 20-35%. This shows that TeleMathLang consistently reduces the number of generated tokens across LLMs and benchmark datasets while maintaining comparable or improved accuracy.

5.1.2 TeleMathLang outperforms concise prompting instructions

Recent prompting strategies such as BeConcise, NoProperGrammar, Chain-of-Draft etc (Lee et al., 2025) have demonstrated the ability to reduce output token count without major loss in accuracy on simpler tasks. Table 3 compares them against TeleMathLang and includes an ablation of TML's keyword design. On GSM8K, a relatively straightforward benchmark, all methods maintain accuracy; on the harder MATH-500, BeConcise, NoProperGrammar, and Chain-of-Draft show degraded accuracy as they reduce step count. TML maintains CoT-level accuracy while achieving 30.4% token reduction.

The ablation on keywords validates our choice to have a set of extended keywords. Removing the extended keywords (TML-Core) drops MATH-500 accuracy by 2.2pp relative to TML-Full, and replacing TML vocabulary with arbitrary structured markers (Random-Keywords) drops it by 3pp, even though the structural format is preserved. A purely generic structure ("Step N:" labels) achieves higher token reduction (34.4%) but at the cost of 3.6pp lower accuracy. The extended keywords allow TML to adaptively increase step count (with keywords such as approach and case, which encourage exploring multiple paths), increasing accuracy in harder problems. All detailed prompt templates are provided in appendix F.

5.1.3 TeleMathLang reduces token count even in models post-trained to generate long output

As mentioned in section 2, recent advancements especially in math benchmarks are largely attributable to significant investments on post-training on CoT datasets. Due to the usage of CoT in SFT, models such as Qwen 2.5 are optimized only in setups where model is asked to generate long step-by-step reasoning. Restricting these models through instructions hampers performance, and we noticed that prompt-based strategies (BeConcise, NoProperGrammar, chain-of-draft) do not work in these models(Table 5).

We finetuned Qwen 2.5 models using pure TeleMathLang samples as well as combined samples for conditional generation, and compared it with CoT prompting, CoT finetuning as well as summarized CoT finetuning. Conditional generation shows the best performance, as the model learns to associate TeleMathLang solutions with CoT since both solutions are present in the training data. We find that finetuning with summarized CoT reduces token count significantly but the reduction comes at a cost of performance. While conditional generation using TeleMathLang is able to match or exceed CoT performance, summarized CoT falls short The results are shown in table 6.

6 Conclusion

In this work, we proposed TeleMathLang, a minimal syntax with reasoning keywords and rules that enables LLMs to generate complete chains-of-thought while reducing total token count. We showed that it generalizes well across instruction-tuned LLMs, and requires light finetuning in models explicitly trained to generate longer chains of thought. TML achieves significant token reduction without compromising accuracy by preserving step-complexity and step-expressiveness — the two properties our theoretical analysis identifies as essential for CoT-level performance. The combination of low resource requirement, strong generalization across models and benchmarks, and compatibility with both prompting and fine-tuning settings positions TeleMathLang as a practical direction for inference-efficient reasoning. Future work could focus on evaluating TML on reasoning-mode models and on further reducing per-step token count.

Limitations

TML prompting has not been evaluated on "reasoning mode" for models such as the Qwen 3 series. The theoretical analysis is suggestive rather than determinative: it provides motivation for why structured compression should work, but step-complexity bounds at the instance level remain empirical hypotheses. The TML keyword set is one principled instantiation of the step-complexity and step-expressiveness requirements; other structured vocabularies satisfying the same functional requirements would likely achieve similar results. We do not compare against latent compression methods (e.g., CCoT, 3TF) that require model retraining; conditional generation is our closest point of contact with training-heavy approaches.

Broader Impact Statement

TeleMathLang reduces inference token cost and latency for LLM reasoning tasks, lowering the computational and financial barrier to deploying accurate reasoning systems. We foresee no direct negative societal impacts from this work. The method does not alter model values or outputs beyond compressing their form; any risks inherit from the underlying LLMs rather than from TML itself.

Author Contributions

Detailed upon deanonymization.

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`.

Anthropic. claude-3-5-sonnet, 2024. URL `https://www.anthropic.com/news/claude-3-5-sonnet`.

Anthropic. Claude sonnet 4.5, 2025. URL https://www.anthropic.com/news/claude-sonnet-4-5.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023. URL https://arxiv.org/abs/2211.12588.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations, 2024. URL https://arxiv.org/abs/2412.13171.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Quy-Anh Dang and Chris Ngo. Reinforcement learning for reasoning in small llms: What works and what doesn't, 2025. URL https://arxiv.org/abs/2503.16219.

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 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, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. Concise reasoning via reinforcement learning, 2025. URL https://arxiv.org/abs/2504.05185.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization, 2022. URL https://arxiv.org/abs/2210.10760.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10: 800–810, 07 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00490. URL https://doi.org/10.1162/tacl_a_00490.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Andreas Hochlehnert, Hardik Bhatnagar, Vishaal Udandarao, Samuel Albanie, Ameya Prabhu, and Matthias Bethge. A sober look at progress in language model reasoning: Pitfalls and paths to reproducibility, 2025. URL https://arxiv.org/abs/2504.07086.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2504.01296.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

Xingshuai Huang, Derek Li, Bahareh Nikpour, and Parsa Omidi. Hierarchical chain-of-thought prompting: Enhancing llm reasoning performance and efficiency, 2026. URL https://arxiv.org/abs/2604.00130.

Amazon Artificial General Intelligence. The amazon nova family of models: Technical report and model card. *Amazon Technical Reports*, 2024. URL https://www.amazon.science/publications/the-amazon-nova-family-of-models-technical-report-and-model-card.

Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. The impact of reasoning step length on large language models, 2024. URL https://arxiv.org/abs/2401.04925.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness, 2024. URL https://arxiv.org/abs/2412.11664.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Ayeong Lee, Ethan Che, and Tianyi Peng. How well do llms compress their own chain-of-thought? a token complexity approach, 2025. URL https://arxiv.org/abs/2503.01141.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems, 2024. URL https://arxiv.org/abs/2402.12875.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers, 2023. URL https://arxiv.org/abs/2207.00729.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz 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, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Qwen. Qwen generation config, 2024. URL https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct/blob/main/generation_config.json.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard,

Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

Canhui Wu, Qiong Cao, Chao Xue, Wei Xi, and Xiaodong He. Efficient reasoning via thought-training and thought-free inference, 2025. URL https://arxiv.org/abs/2511.03408.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less, 2025. URL https://arxiv.org/abs/2502.18600.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning, 2025. URL https://arxiv.org/abs/2502.03387.

A Appendix

B Boolean Circuit Complexity Classes

A **family** of circuits C_n is of constant depth if there exists a constant K such that the depth of C_n is bounded by K for all n. A family of circuits is of polynomial size if there exists a constant c such that the size of C_n is bounded by nc + c for all n. The set AC^0 is the set of families of Boolean circuits of both constant depth and polynomial size (Hao et al., 2022). The set TC^0 is a slightly larger set of circuit families that include threshold gates. These sets, with varying depths, sizes and operations, are used to represent complexity of computations - for instance, an operation like majority voting is too complex for AC^0 but belongs to TC^0.

C TeleMathLang prompt template

TeleMathLang Prompt
<system> Please reason step by step using TeleMathLang, and put your final answer within \boxed{value} <user> You solve math problems using ultra-minimal TeleMathLang language. Here's the syntax syntax: <syntax in table 1> Here's a few examples: <few-shot> Problem: <problem>

Table 7: TeleMathLang Prompt

D Few-shot examples

Here we share a few few-shot TeleMathLang examples. We provide the CoT solution side-by-side **for reference only**. The actual few shot examples included in the prompt **only include the TeleMathLang solutions**.

Example 1: Square Window Problem

Problem: Doug constructs a square window using 8 equal-size panes of glass. The ratio of the height to width for each pane is 5 : 2, and the borders around and between the panes are 2 inches wide. In inches, what is the side length of the square window?

Chain-of-Thought (CoT) Reasoning

1. Identify the dimensions of each pane: Given that the ratio of the height to the width of each pane is $5:2$, let the height of each pane be $5x$ inches and the width be $2x$ inches.

2. Calculate the total dimensions of the window: The window is constructed with 8 panes arranged in 2 rows and 4 columns. The borders between and around the panes are 2 inches wide.

3. Calculate the total width of the window:

- There are 4 panes in each row, each with a width of $2x$ inches.
- There are 3 borders of 2 inches between panes and 2 borders on the sides.
- Total width $= 4(2x) + 5(2) = 8x + 10$

4. Calculate the total height of the window:

- There are 2 panes in each column, each with a height of $5x$ inches.
- There is 1 border between panes and 2 borders at the top and bottom.
- Total height $= 2(5x) + 3(2) = 10x + 6$

5. Set up the equation for the square: Since the window is square,

$$8x + 10 = 10x + 6$$

6. Solve for x:

$$\begin{aligned} 8x + 10 &= 10x + 6 \\ 10 &= 2x + 6 \\ 4 &= 2x \\ x &= 2 \end{aligned}$$

7. Calculate side length:

$$8(2) + 10 = 16 + 10 = 26 \text{ inches}$$

8. Final Answer:

$$\boxed{26}$$

TeleMathLang Reasoning

- Given: Square window, 8 panes, pane ratio 5:2 (h:w), 2-inch borders
- Let: Pane height $= 5x$, width $= 2x$
- Then: Window layout $= 2$ rows $\times$ 4 columns
- Then: Total width $= 4(2x) + 5(2) = 8x + 10$
- Then: Total height $= 2(5x) + 3(2) = 10x + 6$
- Check: $8x + 10 = 10x + 6$
- Then: $x = 2$
- Then: Side length $= 26$
- Verify: $26 = 26$
- **Answer:** $\boxed{26}$

Example 2: Polynomial Interpolation Problem

Problem: Let $P(x)$ be a polynomial of degree $3n$ such that:

$$\begin{aligned} P(3k) &= 2, \\ P(3k-2) &= 1, \\ P(3k-1) &= 0, \quad \text{for } k = 1, \ldots, n \end{aligned}$$

and $P(3n+1) = 730$. Find n.

Chain-of-Thought (CoT) Reasoning

To solve for n, we analyze the polynomial $P(x)$ using Lagrange Interpolation.

1. Define points:

- $P(3k) = 2$ for $k = 0, 1, \ldots, n$
- $P(3k-2) = 1$, $k = 1, \ldots, n$
- $P(3k-1) = 0$, $k = 1, \ldots, n$

2. Interpolation:

$$P(x) = 2\sum_{p=0}^{n} \prod_{\substack{0 \le r \le 3n \\ r \ne 3p}} \frac{x-r}{3p-r} + \sum_{p=1}^{n} \prod_{\substack{0 \le r \le 3n \\ r \ne 3p-2}} \frac{x-r}{3p-2-r}$$

3. Evaluate at $x = 3n+1$:

$$P(3n+1) = 2\sum_{p=0}^{n} \prod_{\substack{0 \le r \le 3n \\ r \ne 3p}} \frac{3n+1-r}{3p-r} + \sum_{p=1}^{n} \prod_{\substack{0 \le r \le 3n \\ r \ne 3p-2}} \frac{3n+1-r}{3p-2-r}$$

4. Simplify with alternating binomial sum:

$$P(3n+1) = \left(\binom{3n+1}{0} - \binom{3n+1}{3} + \binom{3n+1}{6} - \cdots \right) (2(-1)^{3n} - 1) + 1$$

5. Given: $P(3n+1) = 730$**:**

$$(\text{alternating sum})\,(2(-1)^{3n} - 1) = 729$$

6. Use lemma:

$$\frac{2^{3n+2} \cdot \sin^{3n+1}\left(\frac{\pi}{3}\right)}{3} = 729 \Rightarrow n = 4$$

7. Conclusion:

$$\boxed{n = 4}$$

Model	Shot	MATH-500			GSM8K			ARC-Challenge		
		CoT	TML	Tok↓	CoT	TML	Tok↓	CoT	TML	Tok↓
Zero-shot										
Claude 3.5 v1	0	58.6	58.6	57.6%	95.5	95.5	68.1%	95.3	94.7	5.5%
Claude Sonnet 4.5	0	90.8	89.4	30.9%	97.3	97.3	53.3%	97.7	96.2	42.8%
Nova Pro	0	73.8	69.2	37.0%	94.7	94.3	25.8%	93.7	94.3	36.0%
Gemma 3 27B	0	80.0	78.0	35.5%	93.5	93.3	14.9%	93.3	90.0	20.2%
Four-shot										
Claude 3.5 v1	4	51.6	58.6	61.8%	86.6	95.4	71.9%	95.3	95.0	33.3%
Claude Sonnet 4.5	4	90.6	91.0	30.4%	98.3	97.7	41.9%	97.4	98.0	50.5%
Nova Pro	4	70.6	67.4	31.9%	95.4	93.7	57.2%	93.6	94.0	51.0%
Gemma 3 27B	4	77.0	79.0	36.3%	93.7	93.4	37.9%	91.3	92.3	34.7%
Eight-shot										
Claude 3.5 v1	8	51.8	58.8	64.6%	86.9	95.5	67.6%	94.7	94.3	33.6%
Claude Sonnet 4.5	8	91.0	90.8	25.6%	97.7	98.0	42.2%	97.8	98.5	51.5%
Nova Pro	8	74.2	71.4	70.8%	95.2	93.7	78.1%	94.3	92.6	52.0%
Gemma 3 27B	8	72.6	78.6	35.88%	93.1	93.3	26.39%	92.3	92.0	42.5%

Table 8: Per-shot accuracy (%) and output-token reduction for CoT vs. TML across all models and benchmarks.

TeleMathLang Reasoning

Given: $P(x)$ of degree $3n$
Let: $P(3k) = 2, P(3k-2) = 1, P(3k-1) = 0$ for respective ranges
Let: $P(3n+1) = 730$
Approach: Lagrange interpolation
Then:

$$P(x) = 2\sum_{p=0}^{n} \prod_{r \neq 3p} \frac{x-r}{3p-r} + \sum_{p=1}^{n} \prod_{r \neq 3p-2} \frac{x-r}{3p-2-r}$$

So:

$$P(3n+1) = \left(\binom{3n+1}{0} - \binom{3n+1}{3} + \cdots \right) \cdot \left(2(-1)^{3n} - 1\right) + 1$$

Check: Matches given $P(3n+1) = 730$
Insight: Alternating binomial identity
Lemma: $\frac{2^{3n+2} \sin^{3n+1}(\pi/3)}{3} = 729 \Rightarrow n = 4$
Answer: $\boxed{4}$

E Per-Shot Performance Breakdown

Table 8 reports accuracy and Token↓ for each model, benchmark, and shot setting individually. The averaged results are reported in Table 2 in the main paper.

F Baseline Prompt Templates

All baselines use the same system instruction and answer-extraction convention as TML (`\boxed{}`). Below are the exact prompt templates used.

BeConcise (Lee et al., 2025)

BeConcise Prompt

System: Please reason step by step, and put your final answer within \boxed{}.
User: Be concise.
Problem: {problem_text}

NoProperGrammar (Lee et al., 2025)

NoProperGrammar Prompt

System: Please reason step by step, and put your final answer within \boxed{}.
User: Do not use proper grammar.
Problem: {problem_text}

Chain-of-Draft (Xu et al., 2025)

Chain-of-Draft Prompt

System: 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 ####.
User: Problem: {problem_text}

Hi-CoT (Huang et al., 2026)

Hi-CoT Prompt

System: You are a reasoning assistant that solves problems by alternating between `<|instruction|>` (planning what to do next) and `<|execution|>` (executing that plan). Each `<|instruction|>` describes the reasoning step or plan, and each `<|execution|>` performs the corresponding reasoning or computation. You should start with an instruction and follow this format strictly:
`<|instruction|>` Step 1: ...
`<|execution|>` Step 1: ...
`<|instruction|>` Step 2: ...
`<|execution|>` Step 2: ...
...
Finally, put your final answer within \boxed{}.
User: Problem: {problem_text}

TML-Core (8 core keywords only)

TML-Core Prompt

System: Please reason step by step using TeleMathLang, and put your final answer within \boxed{value}.
User: You solve math problems using ultra-minimal TeleMathLang language. Here's the syntax:

Reasoning Keywords: Given (facts and setup), Let (define variables), Then (next logical step), So (intermediate result), Check (verify logic), Therefore (conclusion), Verify (double check), Answer (final result).

Syntax Rules: (1) Begin each line with one of the keywords above. (2) Use LaTeX for mathematical notation. (3) Evaluate expressions when possible.

Problem: {problem_text}
Solution:

Random-Keywords

Random-Keywords Prompt

System: Please reason step by step using the structured language below, and put your final answer within \boxed{value}.
User: You solve math problems using an ultra-minimal structured language. Here's the syntax:

Keywords (you MUST begin each line with one of these): Apple (facts and setup), Blue (define variables), Cat (next logical step), Door (intermediate result), Eagle (verify logic), Frog (conclusion), Gold (double check), Hat (final result), Iron (strategy selection), Jade (case analysis), Kite (key observations), Lamp (for assumptions), Moon (supporting claims), Nest (proof method), Oak (inductive steps), Pine (extending patterns).

Syntax Rules: (1) Begin each line with one of the keywords above. (2) Use LaTeX for mathematical notation. (3) Evaluate expressions when possible (e.g., "Blue $x = 3 \times 2 = 6$").

Problem: {problem_text}
Solution:

Generic-Structured (Step N:)

Generic-Structured Prompt

System: Please reason step by step using the structured language below, and put your final answer within \boxed{value}.
User: You solve math problems using an ultra-minimal structured language. Here's the syntax:

Format: Begin each line with "Step N:" where N is the step number (1, 2, 3, . . .).

Syntax Rules: (1) Begin each line with "Step N:". (2) Use LaTeX for mathematical notation. (3) Evaluate expressions when possible (e.g., "Step 3: $x = 3 \times 2 = 6$").

Problem: {problem_text}
Solution:

G Finetuning Implementation Details

We fine-tune Qwen 2.5 1.5B-Instruct using HuggingFace TRL's `SFTTrainer` with LoRA (Hu et al., 2021). LoRA is applied with standard defaults: rank $r = 8$, $\alpha = 16$, dropout 0.05, targeting all linear layers. Per-device batch size is 8. Training runs on a single GPU and completes in 30–60 minutes per configuration.

Training data. TML solutions are generated using Claude Sonnet 4.5 (Anthropic, 2025). Training problems are drawn from the open-deepscaler dataset (Dang & Ngo, 2025), which does not overlap with the MATH-500, GSM8K, or ARC-Challenge test sets.

Training configurations. We evaluate three fine-tuning settings: (1) TML-only SFT: 10 epochs, 1,000 training examples, learning rate 5×10^{-5}; (2) CoT SFT: same hyperparameters on CoT-format data; (3) Conditional generation (CoT+TML): 5 epochs, 2,000 examples (1,000 CoT + 1,000 TML), learning rate 5×10^{-5}. In the conditional generation setting, each problem appears with both a CoT and a TML solution distinguished by a format-conditioning prefix; at inference, the TML prefix is used to elicit concise output. Inference uses Qwen-recommended settings via vLLM (Qwen, 2024).