Chain-of-Thought Hurts When the Answer Speaks for Itself

Anonymous ACL submission

Abstract

Chain-of-thought (CoT) supervision has become a default assumption in fine-tuning large language models for reasoning tasks, with the intuition being that exposing models to intermediate reasoning steps leads to deeper learning than training on answers alone. We challenge this assumption. We introduce answer *backtrackability*, defined as the degree to which a question can be reconstructed from its answer alone, and argue it is the key determinant of whether CoT supervision helps or hurts during fine-tuning. To test this, we construct two novel benchmarks derived from automata theory (NFA and DFA), where answers are transition tables from which the original question is largely recoverable, and fine-tune three open-source LLMs using LoRA on CoT and No-CoT versions of five datasets: GSM8K, MathInstruct, MATH, and our two formal-language benchmarks. On mathematical datasets, CoT consistently outperforms No-CoT. Within MathInstruct, Program-of-Thought (PoT) problems, where executable code answers are high-backtrackability, favour No-CoT, while CoT-format problems favour CoT. On our formal-language benchmarks, the pattern reverses entirely: **No-CoT outperforms CoT across all models and difficulty tiers. When the answer encodes sufficient structure to reconstruct the reasoning process, intermediate supervision adds noise rather than signal.**

1 Introduction

Large language models (LLMs) have demonstrated remarkable reasoning capabilities across a wide range of tasks, from mathematical problem solving (Wei et al., 2022; Kojima et al., 2022) to logical deduction (Saparov and He, 2023) and code generation (Chen et al., 2021). A central technique driving these gains is CoT prompting and supervision, the practice of providing models with intermediate reasoning steps rather than final answers alone. The intuition is straightforward and accepted: exposing a model to the step-by-step process of reaching a solution encourages deeper learning and more generalizable reasoning, rather than superficial pattern matching over input-output pairs (Wei et al., 2022; Ho et al., 2023).

Yet this intuition has limits that remain poorly understood. Several works have shown that LLMs can behave as sophisticated pattern matchers rather than genuine reasoners, succeeding on benchmarks through surface-level heuristics rather than the reasoning processes their outputs appear to demonstrate (McCoy et al., 2019; Bhattamishra et al., 2020; Dziri et al., 2023). This raises a natural question: *if a model is not truly reasoning but rather learning statistical associations between inputs and outputs, does the presence of intermediate reasoning traces always help — or does it depend on the structure of the task itself?* Even if models *are* genuinely reasoning, does explicit reasoning supervision improve performance?

We argue that the answer depends on a property we introduce called *answer backtrackability*, the degree to which a question can be reconstructed from its answer alone. Consider two contrasting cases, shown in Figure 1. In a grade school arithmetic problem such as *"Janet earns $20/hr and works 52 hrs/week. What is her pay?"*, the answer is a bare integer: $1,040. Given only this number, one cannot recover the original question, as infinitely many problems yield the same answer. The answer carries no trace of the reasoning steps that produced it: the multiplication, the unit conversion, the intermediate products. This is a *low backtrackability* setting, and it is precisely where CoT supervision is most valuable: the explicit reasoning trace preserved in the training signal compensates for what the answer alone cannot convey. Now consider the contrasting case. When a model is asked to *"construct a minimal Deterministic Finite Automata (DFA) over $\{a, b\}$ that accepts all strings*

ending in 'ab' ", the answer is a state transition table listing every transition from every state under every input symbol, together with the designated start and accept states. This table is not merely correct; it is, in a precise sense, *the reasoning itself*. The answer encodes the structure of the solution so completely that the original question is largely recoverable from it. This is a *high backtrackability* setting, where No-CoT supervision is sufficient: the answer already contains the information that a CoT would have made explicit, so forcing the model to generate an additional reasoning trace adds redundancy rather than signal. **We claim that this structural difference, and not task difficulty, domain, or answer length, is the primary determinant of whether CoT supervision helps or hurts during fine-tuning.**

To test this hypothesis, we construct two novel benchmarks derived from automata theory: a Non-Deterministic Finite Automata (NFA) construction dataset and a DFA construction dataset, both with controlled difficulty tiers. We fine-tune three open-source LLMs, Qwen-2.5-7B-Instruct, Qwen-2.5-1.5B-Instruct, and LLaMA-3.1-8B-Instruct, using LoRA on CoT and No-CoT versions of five datasets: GSM8K (Cobbe et al., 2021), MathInstruct (Yue et al., 2024), MATH (Hendrycks et al., 2021), and our two formal-language benchmarks. On all three mathematical datasets, CoT fine-tuning consistently outperforms No-CoT fine-tuning, confirming established intuition. On both formal-language benchmarks, the pattern reverses sharply: No-CoT fine-tuning outperforms CoT across all models and all difficulty tiers, with the gap widening as problem complexity increases.

Our contributions are threefold: (1) we introduce *answer backtrackability* as a principled criterion for predicting when CoT supervision is beneficial; (2) we release two novel formal-language benchmarks with five difficulty tiers and paired CoT/No-CoT splits; and (3) we provide empirical evidence across three model families and five datasets that answer backtrackability reliably predicts the relative utility of CoT versus No-CoT fine-tuning. We release all datasets, fine-tuning code, and model checkpoints at our anonymous repository.

2 Related Work

CoT reasoning has become a central paradigm for improving LLM reasoning performance. Wei et al. (2022) showed that step-by-step reasoning traces

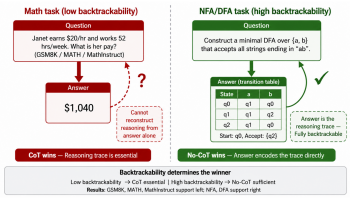

Figure 1: Backtrackability predicts CoT effectiveness.

substantially improve performance on arithmetic and symbolic tasks, with subsequent work extending these gains through zero-shot prompting (Kojima et al., 2022), self-consistency decoding (Wang et al., 2023), and CoT distillation during fine-tuning (Ho et al., 2023). Together, these works established CoT supervision as a widely adopted default for reasoning-oriented training. At the same time, several studies have questioned the universality of these benefits. Sprague et al. (2025) showed that CoT improvements are concentrated primarily in mathematical domains, while Turpin et al. (2023) demonstrated that generated reasoning traces can be systematically unfaithful. Similarly, McCoy et al. (2019) found that transformers often rely on shallow heuristics despite appearing to reason correctly. However, despite these observations, prior work does not provide a principled criterion for predicting when CoT supervision helps or hurts. We propose *answer backtrackability* as such a criterion. Separately, formal language tasks have been widely used to probe the reasoning limits of transformers (Bhattamishra et al., 2020; Shelat et al., 2026). Building on this line of work, we show that supervision format critically affects generalization.

3 Datasets and Experimental Setup

We study two contrasting regimes of answer backtrackability across five datasets: three mathematical benchmarks (low backtrackability) and two formal-language benchmarks we introduce (high backtrackability). For each, we construct paired CoT and No-CoT versions with identical questions.

Mathematical reasoning datasets. We use three established benchmarks. GSM8K (Cobbe et al., 2021) contains $8,500$ grade school arithmetic word problems with integer answers. MathInstruct (Yue et al., 2024) is a curated collection of $262,039$ problems from 13 mathematical reasoning datasets, with hybrid chain-of-thought and program-of-thought (PoT) rationales, where reasoning is expressed as executable code rather than natural language steps. MATH (Hendrycks et al.,

2021) contains $12,500$ competition-level mathematics problems spanning five difficulty levels. For all three, the CoT version uses the full reasoning trace as the target response and the No-CoT version uses only the final answer. In all three cases, answers are bare numbers or expressions from which the original question is not recoverable, a low-backtrackability setting.

Formal-language benchmarks. We introduce two datasets derived from automata theory, both designed so that the answer is a object from which the question is recoverable – a high-backtrackability setting. (i) **NFA dataset.** Each instance consists of a regular expression (RE) and an alphabet; the target answer is the NFA produced by Thompson's construction, presented as a transition table with states, symbol transitions, and ε-transitions. The CoT version includes five intermediate steps: regex parse tree analysis, Thompson's construction rules, NFA state listing, ε-closure computation, and the transition table. The No-CoT version contains only the transition table. We generate $25,000$ instances across five difficulty tiers defined by NFA state count: Tier 1 (4–8 states), Tier 2 (8–14 states), Tier 3 (14–22 states), Tier 4 (22–32 states), and Tier 5 (32–50 states). The test set is stratified with 200 examples per tier. We also construct an out-of-distribution (OOD) test set using paraphrased question templates that do not mention Thompson's construction, to test generalization beyond surface phrasing (Appendices C.3). (ii) **DFA dataset.** Each instance presents a RE and alphabet; the target answer is the minimised DFA produced by Thompson's construction, subset construction, and Hopcroft's minimisation algorithm, represented as a complete transition table. We generate $22,500$ training and $1,000$ test examples across four difficulty tiers defined by minimised DFA state count: Tier 1 (2 states), Tier 2 (2–4 states), Tier 3 (3–6 states), and Tier 4 (4–18 states). The CoT version includes the full construction trace covering every Thompson, subset construction, and Hopcroft refinement step. The No-CoT version contains only the final minimised DFA transition table. Tier 4 is the hardest tier as a error propogates through all downstream states (see Appendix B.2).

Experimental Setup. (i) **Models.** We fine-tune three open-source instruction-tuned LLMs: Qwen-2.5-7B-Instruct, Qwen-2.5-1.5B-Instruct, and LLaMA-3.1-8B-Instruct. We select these models specifically to span two model families and two scales, enabling us to assess whether findings generalize across architectures and parameter counts rather than being artifacts of a single model family. (ii) **Fine-tuning.** All models are fine-tuned using Low-Rank Adaptation (LoRA) (Hu et al., 2022) with rank $r = 32$, $\alpha = 64$, and dropout 0.05, applied to all attention and feed-forward projection layers. We train for 2 epochs with batch size 8, learning rate 2×10^{-4} with cosine decay, and warmup over the first 50 steps. All experiments use a single NVIDIA H100 GPU with bfloat16 precision (see Appendix E.1). (iii) **Evaluation.** For mathematical datasets, we evaluate exact-match accuracy on the final numeric answer with tolerance 10^{-6} for floating-point outputs. For formal-language benchmarks, correctness is evaluated via *structural equivalence* of minimised automata: both the prediction and ground truth are minimised using Hopcroft's algorithm, and a prediction is correct if the resulting automata are isomorphic up to state relabelling. This gives an exact decision over the full infinite language without dependence on string-length bounds. We report accuracy on held-out test sets for all five datasets under both CoT and No-CoT fine-tuning, alongside zero-shot baselines (see Appendix E.2).

4 Results and Analysis

Table 1 reports accuracy across all models, datasets, and training conditions; see Appendix B.

(i) **Mathematical datasets: CoT wins.** On all mathematical benchmarks, CoT fine-tuning outperforms No-CoT across all model sizes. On GSM8K, No-CoT collapses well below the untuned baseline across all models, while CoT yields consistent gains. The collapse is not a formatting artefact: outputs are numerically plausible but fail on multi-step problems requiring intermediate computation, precisely because the answer alone carries no trace of the chained operations that produced it. On MATH-500, the same pattern holds at every difficulty level, with No-CoT degrading most severely at the hardest levels where problems require the longest reasoning chains. Notably, CoT SFT itself degrades all models from their zero-shot baselines on MATH-500, suggesting that the rigid CoT template imposed by SFT interferes with reasoning patterns already present in the pre-trained model, yet CoT SFT still outperforms No-CoT, which degrades further (per-level breakdown in Table 7). On MathInstruct, CoT leads consistently

across all models. The pattern is consistent: when answers are bare numbers from which no reasoning can be recovered, CoT supervision is essential. (ii) **Formal-language benchmarks: No-CoT wins.** On both NFA and DFA datasets, No-CoT outperforms CoT across all models and difficulty tiers. All models score 0.0% at baseline, confirming that these constructions are genuinely out-of-distribution and that fine-tuning gains cannot be attributed to memorisation. The No-CoT advantage grows sharply with tier difficulty: at lower tiers both conditions perform similarly, but at the hardest tiers, where automata have the most states and the reasoning trace is longest, CoT models degrade substantially while No-CoT models remain competitive. This is consistent with the backtrackability hypothesis: at higher complexity the answer table encodes more structure, making the explicit reasoning trace increasingly redundant. The No-CoT advantage also scales with model size, suggesting that larger models better exploit answer structure directly without intermediate traces (Tables 2–5). (iii) **MathInstruct PoT/CoT split.**[1] MathInstruct contains both CoT-format and PoT-format problems. On the PoT subset, No-CoT wins across all models; on the CoT subset, CoT wins across all models. Thus, the same model trained on the same mixture favours different supervision depending on whether the answer structure is recoverable, providing the cleanest within-dataset evidence for the backtrackability hypothesis. (iv) **OOD generalisation.** On the NFA OOD set, Qwen models maintain the No-CoT advantage. LLaMA-3.1-8B No-CoT drops sharply due to parse failures on unseen templates, a format sensitivity failure rather than a reasoning failure. CoT's explicit reasoning structure acts as a format scaffold robust to phrasing variation. On the DFA OOD set, No-CoT maintains its advantage with zero parse failures across all models, consistent with the simpler DFA format being more robust than NFA tables with ε-transitions.

Analysis. (i) **Backtrackability predicts the winner without exception.** Across all 18 model-task pairs, No-CoT wins on high-backtrackability settings while CoT wins on low-backtrackability settings. The MathInstruct PoT/CoT split provides the strongest evidence: the same model trained on the same data mixture switches winner depending on whether the answer encodes the reasoning. (ii)

[1] Program-of-Thought (PoT) problems require generating executable code whose execution yields the answer, making PoT traces highly backtrackable.

Dataset	Model	Base	CoT	No-CoT	Winner
GSM8K	1.5B	50.5	47.0	12.0	CoT
	7B	71.0	**77.0**	17.5	CoT
	8B	85.0	70.0	22.5	CoT
MATH	1.5B	38.0	**24.0**	10.5	CoT
	7B	50.5	**47.0**	24.0	CoT
	8B	43.5	**31.5**	16.0	CoT
MathInstruct	1.5B	15.2	**49.2**	45.8	CoT
	7B	16.6	**63.2**	58.9	CoT
	8B	24.7	**59.3**	54.8	CoT
Overall IID / Hardest tier IID					
NFA	1.5B	0.0 / 0.0	83.5 / 25.0	**86.5 / 47.5**	No-CoT
	7B	0.0 / 0.0	83.4 / 27.0	**88.6 / 52.0**	No-CoT
	8B	0.0 / 0.0	90.3 / 54.5	**93.0 / 69.0**	No-CoT
DFA	1.5B	0.0 / 0.0	79.9 / 53.0	**82.2 / 57.5**	No-CoT
	7B	0.0 / 0.0	88.5 / 71.0	**91.6 / 80.5**	No-CoT
	8B	0.0 / 0.0	90.2 / 74.0	**95.8 / 89.5**	No-CoT

Table 1: Accuracy (%). 1.5B/7B: Qwen-2.5-Instruct; 8B: LLaMA-3.1-Instruct.

The gap grows with difficulty and model size. On formal-language benchmarks, higher tiers require constructing automata with more states, demanding longer and more precise reasoning traces. As complexity increases, CoT models suffer from template rigidity and sequence length pressure while No-CoT models benefit from the richer answer structure at higher tiers. The advantage also scales with model size, suggesting larger models better exploit structured answers directly (Tier 5 failure modes in Appendix D.3). (iii) **CoT SFT can hurt models with strong baselines.** On GSM8K and MATH-500, CoT SFT degrades several models from their zero-shot baselines. The imposed reasoning template conflicts with reasoning strategies already present in the pre-trained model, narrowing the range of approaches available, particularly at the hardest difficulty levels where diverse strategies are most needed. However, even when CoT SFT hurts relative to the baseline, it still outperforms No-CoT SFT, confirming that the issue is format rigidity rather than CoT reasoning itself being harmful.

5 Conclusion

We introduced *answer backtrackability*, defined as the degree to which a question can be reconstructed from its answer, as a criterion for predicting when CoT supervision helps or hurts during fine-tuning. Across five datasets and three model families, CoT outperforms No-CoT on mathematical tasks with information-sparse answers, while No-CoT outperforms CoT on formal-language tasks with structurally rich answers. This challenges the universal benefit of intermediate reasoning supervision.

6 Limitations

While our results consistently support the answer backtrackability hypothesis, several limitations warrant discussion. First, our experimental design places datasets at two extremes of the backtrackability spectrum – bare numeric answers versus full automaton transition tables – and does not systematically study intermediate cases. Whether a continuous relationship exists between the degree of backtrackability and the relative utility of CoT supervision remains an open question for future work.

Second, our formal-language evaluation is limited to two benchmarks from the same domain (automata theory). It is unclear whether the finding generalizes to other structured prediction tasks with high answer backtrackability, such as code generation, semantic parsing, or knowledge graph completion. Broader validation across domains is an important direction for future work.

Third, although we observe that No-CoT fine-tuning outperforms CoT on formal-language benchmarks, we do not provide a mechanistic account of this difference at the level of model internals. Our interpretation – that CoT training encourages template mimicry rather than genuine structural learning – is consistent with the results but not directly verified through probing or interpretability analysis, which we leave for future work.

Finally, all experiments in this work are conducted on open-source instruction-tuned models. Whether the same pattern holds for closed-source models such as GPT-5.1 or Claude remains unknown, as their training data and fine-tuning procedures are not publicly disclosed, making controlled comparison difficult.

7 Ethical Considerations

Data. This work introduces two synthetic benchmarks generated programmatically from regular expressions using a verified algorithmic pipeline. No human subjects are involved, no personally identifiable information is collected or processed, and no crowdsourced annotation is used. The three mathematical datasets used for evaluation – GSM8K, MathInstruct, and MATH – are publicly available benchmarks with established usage in the research community, accessed through their official HuggingFace repositories under their respective licenses. All dataset usage is consistent with the terms of use of the original sources.

Use of AI Assistance. Portions of the writing in this paper were drafted or refined with the assistance of large language model (LLM) tools. All AI-generated or AI-assisted content was reviewed, verified, and edited by the authors, who take full responsibility for the accuracy, integrity, and originality of the work. AI assistance was not used for any experimental design, data generation, model training, or evaluation; all scientific contributions reported in this paper are the product of the authors' own research. The use of AI writing assistance is consistent with the policies of the venue.

Models. All experiments are conducted on publicly released open-source instruction-tuned models: Qwen-2.5-1.5B-Instruct, Qwen-2.5-7B-Instruct, and LLaMA-3.1-8B-Instruct. We do not use any proprietary or closed-source models. All model weights are accessed through their official HuggingFace repositories under their respective licenses.

Environmental Impact. Fine-tuning experiments were conducted on NVIDIA H100 NVL GPUs. Each LoRA fine-tuning run required approximately 1–5 hours of single-GPU compute depending on dataset size, with evaluation runs requiring an additional 1–4 hours per model per dataset. Across all five datasets, three models, and two training conditions (CoT and No-CoT), the total compute budget is estimated at approximately 150–200 GPU hours on H100 hardware. We mitigated environmental impact by using parameter-efficient fine-tuning (LoRA) rather than full model fine-tuning, significantly reducing both memory and compute requirements relative to full fine-tuning at equivalent model scales.

Intended Use and Potential Misuse. The primary contribution of this work is a scientific finding about the conditions under which chain-of-thought supervision is beneficial during fine-tuning. We do not foresee direct harmful applications of this research. However, we acknowledge that insights about when reasoning traces can be omitted from training data could potentially be misused to reduce the transparency of model outputs in high-stakes settings where interpretable step-by-step reasoning is important for accountability or safety.

Reproducibility. All datasets, fine-tuning code, evaluation scripts, and model checkpoints are released publicly to ensure full reproducibility and to support the broader research community.

References

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

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. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14852–14882, Toronto, Canada. Association for Computational Linguistics.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*.

Shlok Shelat, Jay Raval, Souvik Roy, and Manas Gaur. 2026. Beyond memorization: Testing llm reasoning on unseen theory of computation tasks. *Preprint*, arXiv:2601.13392.

Zayne Rea Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. 2025. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*.

A Dataset Examples

This appendix provides examples from all four dataset formats used in our experiments. Each example shows the exact system prompt, user question, and assistant response seen by the model during fine-tuning.

A.1 NFA CoT Example

The following is a complete CoT training example for the regular expression (a|b)*abb over alphabet $\Sigma = \{a, b\}$. The assistant response contains five explicit reasoning steps followed by the final NFA transition table.

System prompt:

You are an expert in formal language theory and automata. When given a regular expression, you construct a Nondeterministic Finite Automaton (NFA) using Thompson's construction. You present the NFA as a clear transition table showing all states, symbol transitions, and epsilon transitions.

User:

Convert the regular expression (a|b)*abb over the alphabet $\Sigma = \{a, b\}$ to an NFA using Thompson's construction. Show the resulting NFA transition table.

Assistant (CoT response):

Steps 1–4: Parse tree and construction trace

```
## Step 1: Analyse the Regular Expression

Given regex: (a|b)*abb
Alphabet Sigma = {a, b}

Parse tree structure:
Concatenation
  Kleene Star (*)
    Union (|)
      Literal 'a'
      Literal 'b'
  Concatenation
    Literal 'a'
    Concatenation
      Literal 'b'
      Literal 'b'

The regex matches any string over {a,b} that
    ↪ ends with 'abb'.
The (a|b)* prefix matches zero or more
    ↪ arbitrary characters,
and 'abb' is the required suffix.

## Step 2: Thompson's Construction Rules
    ↪ Applied

We apply the following Thompson construction
    ↪ rules:

  - Symbol rule for 'a': two states s, a with s
    ↪  --'a'--> a
  - Symbol rule for 'b': two states s, a with s
    ↪  --'b'--> a
  - Union rule: new start q_s with eps-
    ↪ transitions to N(r1).start
    and N(r2).start; both N(r1).accept and N(r2
    ↪ ).accept connect
    via eps to new accept q_a
  - Kleene Star rule: new start q_s and accept
    ↪ q_a;
    q_s --eps--> N(r).start (enter loop)
    q_s --eps--> q_a          (skip, accept
    ↪ empty)
    N(r).accept --eps--> N(r).start (loop back)
    N(r).accept --eps--> q_a         (exit loop)
  - Concatenation rule: build N(r1) and N(r2)
    ↪ independently,
    then add eps-transition from N(r1).accept
    ↪ to N(r2).start

## Step 3: NFA States and Transitions

NFA states: {q1, q2, q3, q4, q5, q6, q7, q8, q9,
    ↪  q10}
Start state : q1
Accept state: q10

Detailed transition listing:

  q1  --eps--> {q2, q7}   [start of (a|b)* and '
    ↪ a' branch]
  q2  --eps--> {q3, q5}   [union: branch to 'a'
    ↪  or 'b']
  q3  --'a'--> {q4}        [symbol 'a' in union
    ↪ ]
  q4  --eps--> {q6}        [exit union via
    ↪ accept]
  q5  --'b'--> {q6}        [symbol 'b' in union
    ↪ ]
  q6  --eps--> {q2, q7}   [loop back or exit
    ↪ star]
  q7  --'a'--> {q8}        [first char of
    ↪ suffix 'abb']
  q8  --'b'--> {q9}        [second char of
    ↪ suffix 'abb']
  q9  --'b'--> {q10}       [third char of
    ↪ suffix 'abb']
  q10: no outgoing transitions (accept sink)

## Step 4: eps-closure of Start State

eps-closure(q1) = {q1, q2, q3, q5, q7}

BFS expansion:
  q1 --eps--> q2  (added q2)
  q1 --eps--> q7  (added q7)
  q2 --eps--> q3  (added q3)
  q2 --eps--> q5  (added q5)
  q3, q5, q7 have no eps-transitions

This set {q1,q2,q3,q5,q7} forms the initial DFA
    ↪  state D0
when processing any input string from the start.
    ↪
```

Step 5: NFA Transition Table. The complete NFA produced by Thompson's construction is shown below. Each row gives all symbol and ε-transitions for one state. $\emptyset$ denotes no transition on that input.

NFA transition table for (a|b)*abb (Step 5 output)

State	Role	a	b	ε
q1	start	$\emptyset$	$\emptyset$	{q2, q7}
q2		$\emptyset$	$\emptyset$	{q3, q5}
q3		{q4}	$\emptyset$	$\emptyset$
q4		$\emptyset$	$\emptyset$	{q6}
q5		$\emptyset$	{q6}	$\emptyset$
q6		$\emptyset$	$\emptyset$	{q2, q7}
q7		{q8}	$\emptyset$	$\emptyset$
q8		$\emptyset$	{q9}	$\emptyset$
q9		$\emptyset$	{q10}	$\emptyset$
q10	accept	$\emptyset$	$\emptyset$	$\emptyset$

```
NFA: 10 states {q1, q2, q3, q4, q5, q6,
            q7, q8, q9, q10}
 Start state:  q1   Accept state:  q10
   Transitions:  9 symbol/eps edges
```

A.2 NFA No-CoT Example

The following is the paired No-CoT training example for the same regular expression (a|b)*abb over $\Sigma = \{a, b\}$. The assistant response contains *only* the final NFA transition table – no parse tree, no construction rules, no state listing, no eps-closure computation.

System prompt:

> *You are an expert in formal language theory and automata. When given a regular expression, you construct a Non-deterministic Finite Automaton (NFA) using Thompson's construction. You present the NFA as a clear transition table showing all states, symbol transitions, and epsilon transitions.*

User:

> Convert the regular expression (a|b)*abb over the alphabet $\Sigma = \{a, b\}$ to an NFA using Thompson's construction. Show the resulting NFA transition table.

The transition table below is identical to the Step 5 output in Section A.1. The CoT example provides four explicit reasoning steps before this table; the No-CoT example provides none. The model trained on the No-CoT condition must learn to produce the correct transition table directly from the question, without any intermediate scaffolding.

Assistant (No-CoT response):

NFA No-CoT assistant response (complete output)

State	Role	a	b	ε
q1	start	$\emptyset$	$\emptyset$	{q2, q7}
q2		$\emptyset$	$\emptyset$	{q3, q5}
q3		{q4}	$\emptyset$	$\emptyset$
q4		$\emptyset$	$\emptyset$	{q6}
q5		$\emptyset$	{q6}	$\emptyset$
q6		$\emptyset$	$\emptyset$	{q2, q7}
q7		{q8}	$\emptyset$	$\emptyset$
q8		$\emptyset$	{q9}	$\emptyset$
q9		$\emptyset$	{q10}	$\emptyset$
q10	accept	$\emptyset$	$\emptyset$	$\emptyset$

```
NFA: 10 states {q1, q2, q3, q4, q5, q6,
          q7, q8, q9, q10}
 Start state:  q1   Accept state:  q10
   Transitions:  9 symbol/eps edges
```

A.3 DFA CoT Example

The following is a complete CoT training example for the regular expression (0|1)*00 over $\Sigma = \{0, 1\}$. The DFA dataset differs from the NFA dataset in two ways: (1) the answer is a minimised DFA requiring three construction phases (Thompson → subset construction → Hopcroft minimisation); (2) the CoT trace is structured as a sequence of JSON objects delimited by TRACE BEGIN and TRACE END markers. The trace shown here is truncated for space; the full training trace records every individual Thompson step, every subset construction state, and every Hopcroft partition refinement iteration.

System prompt:

> *You are a DFA construction assistant. You must output the TRACE and FINAL DFA exactly in the same format as training data. Do not output any explanation outside the TRACE markers.*

User:

> You are given a regular expression over alphabet $\Sigma = \{0, 1\}$. Construct a complete minimised deterministic finite automaton (DFA) that recognizes exactly the same language as the given regular expression. Show all construction steps.
>
> REGEX: (0|1)*00

Assistant (CoT response, truncated for brevity):

DFA CoT assistant response (truncated for brevity)

```
=== TRACE BEGIN ===

### Original Regex
{"regex": "(0|1)*00"}

### Thompson Step (Symbol)
{"token": "0",
 "created_fragment": {"start": "s0", "accept":
    ↪ ["s1"]}}

### Thompson Step (Symbol)
{"token": "1",
 "created_fragment": {"start": "s2", "accept":
    ↪ ["s3"]}}

### Thompson Step (Union) -- constructs (0|1)
{"frag1_start": "s0", "frag1_accepts": ["s1"],
 "frag2_start": "s2", "frag2_accepts": ["s3"],
 "new_start": "s4",   "new_accepts":   ["s5"],
 "eps_from_new_start": ["s0", "s2"],
 "eps_to_new_accept":  ["s1", "s3"]}

### Thompson Step (Star) -- constructs (0|1)*
{"frag_start": "s4", "frag_accepts": ["s5"],
 "new_start":  "s6", "new_accepts":  ["s7"],
 "eps_skip": ["s6","s7"],
 "eps_loop": ["s5","s4"],
 "eps_exit": ["s5","s7"]}

### Thompson Step (Concat) -- constructs (0|1)
    ↪ *0
{"frag1_start": "s6", "frag1_accepts": ["s7"],
 "frag2_start": "s8", "frag2_accepts": ["s9"],
 "new_start": "s6",   "new_accepts": ["s9"],
 "eps_bridge": ["s7","s8"]}

### Thompson Step (Concat) -- constructs (0|1)
    ↪ *00
{"frag1_start": "s6",  "frag1_accepts": ["s9"],
```

```
 "frag2_start": "s10", "frag2_accepts": ["s11"],
    ↪
 "new_start": "s6",    "new_accepts":   ["s11"],
    ↪
 "eps_bridge": ["s9","s10"]}

### Subset Construction Step
{"dfa_state": "D0",
 "nfa_subset": ["s6","s7","s4","s0","s2"],
 "accept": false,
 "transitions": {"0": "D1", "1": "D2"}}

### Subset Construction Step
{"dfa_state": "D1",
 "nfa_subset": ["s1","s5","s7","s4","s0","s2","
    ↪ s8"],
 "accept": false,
 "transitions": {"0": "D3", "1": "D2"}}

### Subset Construction Step
{"dfa_state": "D2",
 "nfa_subset": ["s3","s5","s7","s4","s0","s2"],
 "accept": false,
 "transitions": {"0": "D1", "1": "D2"}}

### Subset Construction Step
{"dfa_state": "D3",
 "nfa_subset": ["s1","s5","s7","s4","s0","s2","
    ↪ s8","s9","s10"],
 "accept": true,
 "transitions": {"0": "D3", "1": "D2"}}

### Hopcroft Initial Partition
{"P": [["D3"], ["D0","D1","D2"]],
 "note": "Accepting vs non-accepting states"}

### Hopcroft Refinement Step 1
{"symbol": "0",
 "splitter": ["D3"],
 "split_block": ["D0","D1","D2"],
 "inter": ["D1","D2"],
 "diff":  ["D0"],
 "new_partition": [["D3"],["D1","D2"],["D0"]]}

### Hopcroft Refinement Step 2
{"symbol": "0",
 "splitter": ["D0"],
 "split_block": ["D1","D2"],
 "inter": ["D1"],
 "diff":  ["D2"],
 "new_partition": [["D3"],["D1"],["D2"],["D0"]]}
    ↪

### Final Partition
{"partition": [["D0"],["D1"],["D2"],["D3"]],
 "note": "All 4 states are distinguishable --
    ↪ DFA is minimal"}

### Renamed States
{"D0": "q0", "D1": "q1", "D2": "q2", "D3": "q3"}
    ↪

=== TRACE END ===

=== FINAL DFA BEGIN ===
states: q0 q1 q2 q3
alphabet: 0 1
start: q0
final: q3
transitions:
q0 0 q1
q0 1 q2
q1 0 q3
q1 1 q2
q2 0 q1
q2 1 q2
q3 0 q3
q3 1 q2
=== FINAL DFA END ===
```

A.4 DFA No-CoT Example

The following is the paired No-CoT training example for the same regular expression (0|1)*00 over $\Sigma = \{0, 1\}$. The assistant response contains only the final minimised DFA transition table – no Thompson steps, no subset construction, no Hopcroft refinement, no JSON trace. The model must produce the correct minimised DFA directly from the regular expression, bypassing the three-phase construction pipeline that the CoT model is explicitly trained to execute step by step.

System prompt:

> *You are a DFA construction assistant. You must output ONLY the DFA in the exact required format. Do not provide any explanation, extra text, or commentary.*

User:

> You are given a regular expression over alphabet $\Sigma = \{0, 1\}$. Construct a complete minimised deterministic finite automaton (DFA) that recognizes exactly the same language as the given regular expression.
>
> Output format (strict):
> states: q0 q1 q2 ...
> alphabet: 0 1
> start: q0
> final: qX qY ...
> transitions:
> q0 0 q1
> q0 1 q2
> ...
>
> Constraints: (1) DFA must be complete – a transition must be defined for every state-symbol pair. (2) DFA must be deterministic – exactly one transition per state-symbol pair. (3) Use only q0, q1, q2, ... naming. (4) Output the final DFA only. No explanation.
>
> REGEX: (0|1)*00

Assistant (No-CoT response):

DFA No-CoT assistant response

```
=== FINAL DFA BEGIN ===
states: q0 q1 q2 q3
alphabet: 0 1
```

```
start: q0
final: q3
transitions:
q0 0 q1
q0 1 q2
q1 0 q3
q1 1 q2
q2 0 q1
q2 1 q2
q3 0 q3
q3 1 q2
=== FINAL DFA END ===
```

The minimised DFA for (0|1)*00 has four states, each corresponding to a distinct suffix of the input seen so far: q_0 is the start state (no relevant suffix seen); q_1 records that the last symbol was 0; q_3 is the accepting state (the last two symbols were 00); and q_2 is a reset state (last symbol was 1, so any partial suffix is lost). As with the NFA examples, the final answer is identical in Sections A.3 and A.4 – the only difference is whether the model is trained to produce the construction trace before it.

B Full Results

This appendix reports complete per-tier accuracy breakdowns for all models, datasets, and evaluation conditions. All results are confirmed from experimental logs. Bold indicates the better result between CoT SFT and No-CoT SFT for each row. All values are accuracy percentages on held-out test sets.

The three models evaluated are Qwen-2.5-1.5B-Instruct (1.5B), Qwen-2.5-7B-Instruct (7B), and LLaMA-3.1-8B-Instruct (8B), spanning two model families and two capacity scales. The two training conditions are CoT SFT (fine-tuned on full reasoning traces) and No-CoT SFT (fine-tuned on final answers only). Baseline denotes the zero-shot performance of the unmodified pre-trained model with no fine-tuning applied.

B.1 NFA Dataset: Full Tier Breakdown

The NFA dataset has five difficulty tiers defined by NFA state count: T1 (4–8 states), T2 (8–14), T3 (14–22), T4 (22–32), and T5 (32–50). The test set contains 200 examples per tier (1,000 total). We report results under two evaluation conditions:

- **IID**: questions use the same five templates seen during training, all of which explicitly mention Thompson's construction. This measures how well the fine-tuned model has learned the construction task under familiar question phrasing.
- **OOD**: questions use eight paraphrased templates not seen during training, none of which mention Thompson's construction by name. This tests whether fine-tuned models have learned the underlying construction task or have overfit to surface phrasing cues in the training questions.

Table 2: NFA dataset – IID test accuracy (%) per tier. Bold = better of CoT / No-CoT. All results confirmed.

Train	Model	Overall	T1	T2	T3	T4	T5
Baseline	1.5B	0.0	0.0	0.0	0.0	0.0	0.0
	7B	0.0	0.0	0.0	0.0	0.0	0.0
	8B	0.0	0.0	0.0	0.0	0.0	0.0
CoT SFT	1.5B	83.5	**100.0**	**100.0**	97.5	**94.0**	25.0
	7B	83.4	**100.0**	**100.0**	97.5	**97.0**	27.0
	8B	90.3	**100.0**	**100.0**	99.5	**98.5**	54.5
No-CoT SFT	1.5B	**86.5**	**100.0**	**100.0**	**98.5**	93.5	**47.5**
	7B	**88.6**	**100.0**	**100.0**	**99.0**	95.0	**52.0**
	8B	**93.0**	**100.0**	**100.0**	**100.0**	**98.5**	**69.0**

Table 3: NFA dataset – OOD test accuracy (%) per tier. Bold = better of CoT / No-CoT. LLaMA-8B No-CoT OOD includes 118 parse failures (12.5% of the test set).

Train	Model	Overall	T1	T2	T3	T4	T5
Baseline	1.5B	0.0	0.0	0.0	0.0	0.0	0.0
	7B	0.0	0.0	0.0	0.0	0.0	0.0
	8B	0.0	0.0	0.0	0.0	0.0	0.0
CoT SFT	1.5B	80.4	89.0	**95.0**	**94.0**	**93.0**	24.0
	7B	80.6	91.0	**97.0**	**96.0**	**96.0**	26.0
	8B	**89.3**	**100.0**	**98.0**	**96.5**	**95.5**	58.0
No-CoT SFT	1.5B	**81.5**	**95.0**	90.0	91.0	88.0	**48.0**
	7B	**84.0**	**97.2**	92.5	93.5	90.0	**50.5**
	8B	80.5	82.9	80.0	83.5	82.5	**74.0**

IID observations. No-CoT SFT outperforms CoT SFT on overall IID accuracy for all three model sizes, with the gap largest at Tier 5 – the hardest tier with 32–50 NFA states. The T5 No-CoT advantage is 22.5 pp for Qwen-7B (52.0% vs. 27.0%), 14.5 pp for LLaMA-8B (69.0% vs. 54.5%), and 22.5 pp for Qwen-1.5B (47.5% vs. 25.0%). This is the strongest single result supporting the backtrackability hypothesis: at the highest complexity level, CoT supervision actively limits the model's ability to produce correct outputs, while No-CoT forces the model to internalise the full mapping directly.

At Tier 4, an interesting anomaly appears for the two Qwen models: CoT SFT achieves 94.0% (1.5B) and 97.0% (7B) while No-CoT achieves 93.5% and 95.0% respectively, making

CoT marginally better at T4 despite being substantially worse overall. LLaMA-8B does not exhibit this: No-CoT ties CoT at T4 (98.5% each) and wins decisively at T5. The T4 reversal for Qwen models likely reflects that T4 NFAs (22–32 states) are structurally complex enough to benefit from the CoT reasoning scaffold but still within its scope, while T5 NFAs (32–50 states) exceed the template's capacity entirely.

At Tiers 1 and 2, both conditions achieve 100% for all three models, confirming that small NFAs are trivially solved by any fine-tuned model regardless of training condition. Tier 3 shows a small No-CoT advantage (e.g. 99.0% vs. 97.5% for Qwen-7B).

OOD observations. The OOD results reveal an important asymmetry between model families. For Qwen-1.5B and Qwen-7B, No-CoT maintains its overall advantage under OOD evaluation (Qwen-7B: 84.0% vs. 80.6%), but CoT SFT outperforms No-CoT SFT at mid-complexity tiers T2–T4. For example, Qwen-7B CoT achieves 97.0% at T2 versus No-CoT's 92.5%, and 96.0% at T4 versus No-CoT's 90.0%. This tier-level reversal under OOD conditions suggests that No-CoT models at mid-complexity levels partially rely on surface question features to guide construction – when the question phrasing changes, No-CoT performance drops more than CoT performance at these tiers. No-CoT maintains its advantage at T1 and T5, suggesting that at the simplest and hardest extremes, surface phrasing matters less than the structural richness of the answer.

LLaMA-8B presents a starkly different OOD profile. CoT SFT achieves 89.3% overall OOD while No-CoT SFT achieves only 80.5%, a reversal from the IID result where No-CoT won (93.0% vs. 90.3%). This reversal is driven entirely by 118 parse failures (12.5% of the OOD test set) in the No-CoT condition – the model outputs garbled or incorrectly formatted transition tables on unseen question templates, not wrong answers. This is a format sensitivity failure rather than a reasoning failure: LLaMA No-CoT learned to rely on surface phrasing cues in the training templates to structure its output, and when those cues are absent (as in OOD templates that do not mention Thompson's construction by name), the output format breaks down. CoT's explicit reasoning structure – which begins with parse tree analysis regardless of how the question is phrased – acts as a format scaffold that insulates the model from this collapse. Notably, even under this collapse, No-CoT still wins at T5 OOD (74.0% vs. 58.0%), where the structural richness of the answer is most extreme.

The IID-to-OOD accuracy drop is 2.9 pp for CoT and 4.6 pp for No-CoT overall on Qwen-7B, indicating that CoT models generalise slightly better to unseen question formats. This is consistent with CoT's explicit reasoning structure being invariant to surface phrasing changes.

B.2 DFA Dataset: Full Tier Breakdown

The DFA dataset has four difficulty tiers defined by minimised DFA state count: T1 (2 states), T2 (2–4), T3 (3–6), T4 (4–18). The test set contains 1000 examples across the four tiers. Tier 4 is the most challenging because a single error in an early Thompson or subset construction step propagates through all downstream states, and Hopcroft minimisation then produces a structurally different automaton. We evaluate under both IID and OOD conditions using the same template split as the NFA dataset.

Table 4: DFA dataset – IID test accuracy (%) per tier. Bold = better of CoT / No-CoT. All results confirmed.

Train	Model	Overall	T1	T2	T3	T4
Baseline	1.5B	0.0	0.0	0.0	0.0	0.0
	7B	0.0	0.0	0.0	0.0	0.0
	8B	0.0	0.0	0.0	0.0	0.0
CoT SFT	1.5B	79.9	**100.0**	96.5	86.0	53.0
	7B	88.5	**100.0**	**98.5**	93.5	71.0
	8B	90.2	**100.0**	99.0	95.5	74.0
No-CoT SFT	1.5B	**82.2**	**100.0**	**97.0**	**88.5**	**57.5**
	7B	**91.6**	**100.0**	98.0	**94.5**	**80.5**
	8B	**95.8**	**100.0**	**99.5**	**97.5**	**89.5**

Table 5: DFA dataset – OOD test accuracy (%) per tier. Bold = better of CoT / No-CoT. All results confirmed. LLaMA-8B CoT OOD had 46 parse failures (7.2%); No-CoT had zero parse failures across all models.

Train	Model	Overall	T1	T2	T3	T4
Baseline	1.5B	0.0	0.0	0.0	0.0	0.0
	7B	0.0	0.0	0.0	0.0	0.0
	8B	0.0	0.0	0.0	0.0	0.0
CoT SFT	1.5B	71.0	92.7	82.5	78.0	48.0
	7B	87.5	**100.0**	**96.5**	**94.0**	69.5
	8B	82.4	82.9	81.5	84.0	81.5
No-CoT SFT	1.5B	**76.6**	**90.2**	**87.5**	**83.0**	**56.5**
	7B	**90.3**	**100.0**	**96.5**	93.0	**79.5**
	8B	**88.3**	**100.0**	**91.0**	**88.0**	**83.5**

IID observations. No-CoT SFT outperforms CoT SFT overall and at Tier 4 for all three models. The Tier 4 IID gaps are 4.5 pp for Qwen-1.5B (57.5% vs. 53.0%), 9.5 pp for Qwen-7B (80.5% vs. 71.0%), and 15.5 pp for LLaMA-8B (89.5% vs. 74.0%). This is a clean scaling pattern: the larger the model, the more it benefits from No-CoT supervision on high-backtrackability tasks. LLaMA-8B No-CoT achieves 95.8% overall IID, the highest result in the entire DFA experiment, and 89.5% at T4, which is 15.5 pp above CoT.

At Tier 1, all models reach 100% under both conditions, confirming that two-state DFAs are trivially solved. At T2, results are near-perfect for all models. At T3, No-CoT wins for all three models (88.5% vs. 86.0% for 1.5B; 94.5% vs. 93.5% for 7B; 97.5% vs. 95.5% for LLaMA-8B). The No-CoT advantage is consistent in direction across all tiers and all models, with no exceptions.

OOD observations. No-CoT maintains its overall OOD advantage for all three models (Qwen-1.5B: 76.6% vs. 71.0%; Qwen-7B: 90.3% vs. 87.5%; LLaMA-8B: 88.3% vs. 82.4%). No-CoT records zero parse failures across all models on the DFA OOD set; LLaMA-8B CoT suffers 46 parse failures (7.2%), driving its OOD drop of 7.8 pp (90.2% $\rightarrow$ 82.4%). This contrasts with the NFA OOD results where it is No-CoT that collapses (118 parse failures for LLaMA-8B). The simpler, more standardised output format of minimised DFA transition tables (compared to NFA tables with ε-transitions) appears to be inherently more robust to question template variation for No-CoT models, while CoT models for LLaMA retain format sensitivity regardless of dataset. LLaMA-8B No-CoT OOD (88.3%) still comfortably exceeds CoT OOD (82.4%), confirming No-CoT's advantage is robust even accounting for the CoT parse failures.

At Tier 4 OOD, No-CoT wins for all three models: 56.5% vs. 48.0% (1.5B), 79.5% vs. 69.5% (7B), 83.5% vs. 81.5% (LLaMA-8B). The hardest tier is also the most robust to template variation for No-CoT models, consistent with the structural richness of the answer being the dominant learning signal at high complexity.

B.3 Mathematical Datasets: Full Results

Table 6 reports accuracy on all three mathematical reasoning benchmarks for all models and training conditions. Unlike the formal-language benchmarks, these datasets do not have difficulty tiers defined by structural complexity; we report overall accuracy. For MATH-500 we also report per-level breakdowns in Table 7 since the dataset defines five competition difficulty levels (L1–L5).

Table 6: Mathematical datasets – overall accuracy (%). Bold = better of CoT / No-CoT. All results confirmed.

Train	Model	GSM8K	MATH-500	MathInstruct
Baseline	1.5B	50.5	38.0	15.2
	7B	71.0	50.5	16.6
	8B	85.0	43.5	24.7
CoT SFT	1.5B	**47.0**	**24.0**	**49.2**
	7B	**77.0**	**47.0**	**63.2**
	8B	**70.0**	**31.5**	**59.3**
No-CoT SFT	1.5B	12.0	10.5	45.8
	7B	17.5	24.0	58.9
	8B	22.5	16.0	54.8

Table 7: MATH-500 – per-level accuracy (%) for Qwen-2.5-7B. L1 (easiest) to L5 (hardest). Bold = better of CoT / No-CoT.

Train	Model	L1	L2	L3	L4	L5
Baseline	7B	81.2	48.7	60.5	54.7	28.6
CoT SFT	7B	**81.2**	**64.1**	**58.1**	**47.2**	**12.2**
No-CoT	7B	68.8	43.6	23.3	13.2	6.1

Table 8: MathInstruct PoT and CoT subtask breakdown (%). No-CoT wins on PoT, while CoT wins on the CoT subtask across all models. Bold indicates the better result within each subtask.

Model	PoT-C	PoT-N	CoT-C	CoT-N
1.5B	65.5	**66.2**	**43.1**	38.2
7B	69.1	**70.9**	**61.0**	54.4
8B	67.6	**71.0**	**56.2**	48.8

GSM8K observations. The No-CoT collapse on GSM8K is the most severe finding in the mathematical results. All three models fall to 12.0–22.5% under No-CoT SFT, well below the untuned zero-shot baseline (50.5–85.0%). This means No-CoT fine-tuning actively degrades the model's arithmetic reasoning ability compared to no fine-tuning at all. Models produce numerically plausible outputs but fail on problems requiring multiple intermediate steps, because No-CoT training provides no signal about how intermediate quantities should be computed and chained. The collapse is consistent across all three model sizes, with LLaMA-8B No-CoT (22.5%) showing the least severe drop despite having the highest baseline (85.0%) – a 62.5 pp gap between baseline and No-CoT SFT.

CoT SFT improves over baseline for Qwen-7B (71.0% $\rightarrow$ 77.0%) but falls below baseline

for Qwen-1.5B (50.5% → 47.0%) and LLaMA-8B (85.0% → 70.0%). The LLaMA-8B drop of 15 pp is particularly striking: a strong pre-trained model is harmed by CoT SFT because the imposed reasoning template conflicts with its native reasoning patterns. Despite this, CoT SFT still outperforms No-CoT SFT by a massive margin (70.0% vs. 22.5% for LLaMA-8B), confirming that the issue is format rigidity, not CoT reasoning being ineffective.

MATH-500 observations. CoT SFT outperforms No-CoT SFT at every difficulty level for Qwen-7B (Table 7). The No-CoT degradation is most extreme at the harder levels: L3 (23.3% vs. 58.1%), L4 (13.2% vs. 47.2%), and L5 (6.1% vs. 12.2%). No-CoT completely collapses at L5 for Qwen-1.5B (0.0%). Notably, CoT SFT itself falls below the baseline at every level for all models – for instance, Qwen-7B baseline achieves 81.2% at L1 and 28.6% at L5, while CoT SFT achieves 81.2% at L1 and only 12.2% at L5. This suggests that SFT on competition mathematics data imposes a format constraint that narrows the model's range of reasoning strategies, particularly at the hardest levels where diverse approaches are most needed. Yet CoT SFT always dominates No-CoT SFT, confirming that even degraded CoT supervision is far better than no supervision on low-backtrackability tasks.

MathInstruct observations. The PoT/CoT split within MathInstruct is the most direct within-dataset test of the backtrackability hypothesis. On the PoT subset, where program code outputs encode the full reasoning trace and are therefore high-backtrackability answers, No-CoT wins across all three models: 66.2% vs. 65.5% (1.5B), 70.9% vs. 69.1% (7B), 71.0% vs. 67.6% (LLaMA-8B). On the CoT-format subset, where natural language reasoning steps are not recoverable from the final answer and the setting is low-backtrackability, CoT SFT wins across all three models: 43.1% vs. 38.2% (1.5B), 61.0% vs. 54.4% (7B), 56.2% vs. 48.8% (LLaMA-8B). Both effects scale with model capacity: the PoT No-CoT advantage grows from 0.7 pp (1.5B) to 1.8 pp (7B) to 3.4 pp (LLaMA-8B), and the CoT-subtask CoT advantage grows from 4.9 pp (1.5B) to 6.6 pp (7B) to 7.4 pp (LLaMA-8B). This is the cleanest evidence in the paper that the relevant variable is the structure of the answer, not the dataset or model identity.

C Dataset Construction Details

This appendix describes the full generation pipelines for the NFA and DFA datasets, including tier definitions, operator weight distributions, state count bounds, verification procedures, and the complete set of OOD question templates used for out-of-distribution evaluation.

C.1 NFA Generation Pipeline

Each NFA dataset example is generated through the following five-stage pipeline.

Stage 1: Alphabet selection. An alphabet is sampled uniformly from five predefined options: binary $\{0, 1\}$, ab $\{a, b\}$, abc $\{a, b, c\}$, abcd $\{a, b, c, d\}$, and digits $\{0, 1, 2, 3\}$.

Stage 2: Regex AST generation. A random regular expression is sampled as an abstract syntax tree (AST) using a tiered generator. The generator supports the following operators: concatenation, union ($|$), Kleene star ($*$), plus ($+$), optional ($?$), and composite patterns including star_union ($(X|Y)^*$), star_concat ($(XY)^*$), star_star ($(X^*)^*$), union_chain ($X|Y|Z$), concat3 (XYZ), and concat4 ($XYZW$). Sugar operators ($+$, $?$) are desugared to core operators before construction: $r^+ \to rr^*$ and $r? \to r|\varepsilon$. The generator is parameterised by tier, which controls the maximum recursion depth, the base leaf probability (the probability of terminating recursion at any node), and the operator weight distribution.

Stage 3: Thompson's NFA construction. The desugared AST is converted to an NFA using Thompson's construction, which guarantees a single start state and a single accept state. The construction is recursive over the AST structure, applying the following rules: (1) a literal c produces two states s, a with $s \xrightarrow{c} a$; (2) concatenation $r_1 r_2$ connects $N(r_1).\mathit{accept}$ to $N(r_2).\mathit{start}$ via ε; (3) union $r_1|r_2$ adds a new start with ε-transitions to both sub-NFAs and a new accept reachable via ε from both sub-NFA accepts; (4) Kleene star r^* adds a new start s and accept a, with four ε-transitions: $s \to N(r).\mathit{start}$ (enter), $s \to a$ (skip, accepting the empty string), $N(r).\mathit{accept} \to N(r).\mathit{start}$ (loop back), and $N(r).\mathit{accept} \to a$ (exit after one or more iterations).

Stage 4: Verification. The produced NFA is verified by cross-checking it against a DFA built via subset construction on all strings up to a tier-dependent maximum length (6–10 characters depending on tier). Any example where the con-

Table 9: NFA dataset tier definitions. Leaf prob denotes the base probability of terminating recursion at a leaf node.

Tier	State range	Max depth	Leaf prob	Train %	Description
T1	4–8	3	0.55	10	concat + union only
T2	8–14	4	0.42	20	+ star, plus, optional
T3	14–22	5	0.30	25	+ star_union, star_concat
T4	22–32	6	0.22	25	+ concat4, nested closures
T5	32–50	8	0.15	20	+ star_star, union_chain

Table 10: Operator sampling weights by tier (normalized to sum to 1.0). Operators not listed for a tier are unavailable. The final column combines `star_star` and `union_chain`, available at T5 only.

Tier	concat	union	star	plus	optional	concat3	concat4	star_ union	star_ concat	star_star union_chain
T1	0.55	0.35	–	–	–	0.10	–	–	–	–
T2	0.28	0.20	0.22	0.14	0.08	0.08	–	–	–	–
T3	0.22	0.16	0.20	0.12	–	0.10	–	0.12	0.08	–
T4	0.18	0.14	0.20	0.10	–	0.10	0.08	0.12	0.08	–
T5	0.14	0.10	0.18	0.10	–	0.08	0.08	0.10	0.08	0.14

structed NFA and a reference DFA (built via subset construction solely for verification purposes) disagree on any test string is discarded. Examples are also deduplicated by an MD5 hash of the regex string and alphabet pair.

Stage 5: Formatting. Each verified example is formatted into either a CoT trace (Steps 1–5 followed by the NFA transition table) or a No-CoT response (the transition table only), as shown in Appendix A.1 and A.2.

Tier definitions. The five tiers are defined by NFA state count bounds and generator parameters. Table 9 gives the full specification.

Operator weight distributions. Table 10 gives the operator sampling weights for each tier. Higher tiers assign greater weight to closure operators (star, star_union, star_concat) and longer concatenation chains (concat3, concat4), which produce larger NFAs with more epsilon transitions. **Dataset statistics.** The full NFA dataset contains 25,000 unique examples across all five tiers. The test set is stratified with exactly 200 examples per tier (1,000 total). The remaining examples are split 90% train / 5% validation / 5% held-out, with shuffling seeded at 42 for reproducibility. The dataset additionally includes 26 handcrafted examples covering canonical textbook regular expressions such as $(a|b)^*abb$, $(0|1)^*00$, and $(ab|ba)^*$, ensuring coverage of well-known patterns that random generation may under-represent.

C.2 DFA Generation Pipeline

The DFA dataset generation pipeline shares Stages 1–2 with the NFA pipeline (alphabet selection and regex AST generation) but replaces Stage 3 with a three-phase construction and adds a more stringent verification step.

Stage 3: Full DFA construction pipeline. The regex AST is converted to a minimised DFA via three sequential phases:

(1) **Thompson's construction**: produces an NFA with a single start and accept state, as described in Appendix C.1.

(2) **Subset (powerset) construction**: converts the NFA to a DFA by tracking sets of NFA states. Starting from the ε-closure of the NFA start state, new DFA states are created for each reachable ε-closure under each input symbol. A DFA state is accepting if it contains the NFA accept state.

(3) **Hopcroft's minimisation**: partitions DFA states into equivalence classes of indistinguishable states. The initial partition separates accepting from non-accepting states plus a virtual dead/trap state. The worklist algorithm iteratively refines the partition by splitting blocks on each input symbol until no further splits are possible. Dead states are removed from the final minimised DFA.

Stage 4: Verification. The NFA, intermediate DFA, and minimised DFA are all cross-verified by simulating each on all strings up to length 7 and confirming $L(\text{NFA}) = L(\text{DFA}) = L(\text{minDFA})$. Any example failing any pairwise check is discarded. This three-way check is more stringent than the NFA dataset's two-way check and catches both construction errors and minimisation regressions.

Stage 5: Formatting. Each verified example is formatted into either a CoT trace (JSON-structured TRACE block followed by a FINAL DFA block, as shown in Appendix A.3) or a No-CoT response (FINAL DFA block only, as shown in Appendix A.4).

Tier definitions. The four DFA tiers are defined by minimised DFA state count and generator parameters. Unlike the NFA tiers which measure raw NFA states, DFA tiers measure the minimised state count, which reflects the true structural complexity of the recognised language after all redundant states are removed.

Dataset statistics. The full DFA dataset contains 22,500 training examples and 1,000 test examples,

Table 11: DFA dataset tier definitions.

Tier	State range	Max DFA states	Train %	Description
T1	2	24	15	direct symbol mapping, simple concat/union
T2	2–4	24	25	Kleene star, plus, optional introduced
T3	3–6	24	35	combined operators, star-of-union patterns
T4	4–18	24	25	deeply nested closures, full subset construction

split 90% train / 5% IID validation / 5% OOD validation. The dataset includes 44 handcrafted examples covering classic textbook patterns. A maximum pre-minimisation DFA state count of 24 and a maximum minimised DFA state count of 18 are enforced; examples exceeding these bounds are discarded to keep sequence lengths tractable.

C.3 OOD Question Templates

The NFA OOD evaluation uses eight question templates that differ structurally from the five training templates in the following ways: (1) no template mentions Thompson's construction by name; (2) templates use varied framing including informal, homework-problem, implementation, and formal mathematical styles; (3) regex is presented without backtick formatting; (4) some templates ask for specific components (epsilon transitions, state names) rather than generically asking for a transition table. This tests whether fine-tuned models have learned the underlying NFA construction task or have overfit to the surface phrasing of training questions.

The five training templates all follow the pattern: *"[verb] the regular expression 'regex' over the alphabet $\Sigma = \{\ldots\}$ [to/using] Thompson's [NFA] construction. [Show/Give/Present] the [resulting] NFA transition table."* All five explicitly name Thompson's construction and use backtick-formatted regex.

The eight OOD templates are as follows. Each is shown with a representative instantiation for regex $(a|b)^*abb$ over $\Sigma = \{a, b\}$.

OOD Template 1 (informal):

> I need an NFA for the pattern $(a|b)^*abb$ using symbols $\{a, b\}$. Please show me the full state transition table with epsilon moves.

OOD Template 2 (homework-problem style):

> Design a nondeterministic finite automaton that recognizes the language defined by $(a|b)^*abb$ over $\{a, b\}$. List all states, mark which is the start state and which are accepting, and give the complete transition table.

OOD Template 3 (reverse framing):

> What NFA would you build to recognize strings matching $(a|b)^*abb$ where the alphabet is $\{a, b\}$? Show the transition table.

OOD Template 4 (no backticks, component focus):

> For the regular expression $(a|b)^*abb$ with input alphabet $\{a, b\}$, construct a finite automaton with nondeterminism. Present the states and how they transition on each symbol and on epsilon.

OOD Template 5 (textbook exercise style):

> Exercise: Given $r = (a|b)^*abb$ over alphabet $\{a, b\}$, draw (as a table) the NFA M such that $L(M) = L(r)$. Clearly indicate the start state and final state.

OOD Template 6 (implementation framing):

> I want to implement a regex matcher for $(a|b)^*abb$ on the alphabet $\{a, b\}$. Build me the NFA state machine – show every state, every transition on input symbols, and every epsilon transition.

OOD Template 7 (verification framing):

> Suppose we want to check if strings over $\{a, b\}$ match the pattern $(a|b)^*abb$. What NFA would do this? Give the complete transition function as a table.

OOD Template 8 (formal mathematical notation):

> Let $r = (a|b)^*abb$ be a regular expression over $\Sigma = \{a, b\}$. Construct an NFA $N = (Q, \Sigma, \delta, q_0, F)$ such that $L(N) = L(r)$. Specify Q, q_0, F, and δ in tabular form.

The OOD templates span a wide range of phrasings, from informal first-person requests (T1, T6) to formal mathematical notation (T8). None uses backtick regex formatting. None names Thompson's construction. Templates T2, T5, and T8 use formal academic language but frame the task differently

from the training templates (homework, exercise, and formal specification respectively). Templates T3 and T7 use a reverse or verification framing that implicitly asks the model to think about what the NFA *does* rather than how to *construct* it.

Each OOD test example uses the same regex, alphabet, and ground-truth NFA as its paired IID example, with only the question template changed. This ensures that any performance difference between IID and OOD evaluation is attributable solely to question phrasing and not to differences in task difficulty.

D Error Analysis

This appendix presents concrete examples of the failure modes observed in our experiments. Examples in Sections D.1 and D.3 are illustrative – constructed to be consistent with observed failure patterns from model output inspection. Examples in Section D.2 are drawn directly from confirmed evaluation logs.

D.1 CoT Fails, No-CoT Succeeds: Template Mimicry

At Tier 5 of the NFA dataset, CoT SFT achieves only 27.0% accuracy while No-CoT SFT achieves 52.0% – a gap of 25 percentage points. Inspection of CoT model outputs at this tier reveals a consistent failure pattern we term *template mimicry*: the model produces all five reasoning steps with correct formatting and plausible-looking content, but the final transition table is wrong. The intermediate steps appear coherent in isolation but do not correctly implement Thompson's construction for the given regex. By contrast, the No-CoT model, which is never trained to produce intermediate steps, outputs the transition table directly and correctly.

The following illustrative example demonstrates this failure mode for a Tier 5 regex ((a|b)*ab(ba)*)+ over $\Sigma = \{a, b\}$ (NFA state count: 38).

Ground truth NFA (partial, key states shown):

Ground truth – selected transitions

State	Role	a	b	ε
q1	start	$\emptyset$	$\emptyset$	{q2, q19}
q2		$\emptyset$	$\emptyset$	{q3, q5}
q3		{q4}	$\emptyset$	$\emptyset$
q4		$\emptyset$	$\emptyset$	{q6}
q5		$\emptyset$	{q6}	$\emptyset$
q6		$\emptyset$	$\emptyset$	{q2, q7}
q7		{q8}	$\emptyset$	$\emptyset$
q8		$\emptyset$	{q9}	$\emptyset$
...	...	...	...	...
q38	accept	$\emptyset$	$\emptyset$	{q2, q19}

CoT model output (illustrative):

The CoT model correctly identifies the parse tree structure in Step 1 and correctly lists the applicable Thompson rules in Step 2. However, in Step 3 the model begins to conflate states from different sub-NFAs, incorrectly assigning epsilon transitions that belong to the $(ba)^*$ sub-NFA to states in the $(a|b)^*$ sub-NFA. The error propagates silently through Steps 4 and 5, producing a transition table that looks well-formed but represents a different language than the input regex. The model's Step 3 contains the following incorrect transitions:

CoT model – Step 3 output (incorrect transitions highlighted)

```
## Step 3: NFA States and Transitions

NFA states: {q1, q2, ..., q38}
Start state : q1
Accept state: q38

  q1  --eps--> {q2, q19}    [correct]
  q2  --eps--> {q3, q5}     [correct]
  q3  --'a'--> {q4}          [correct]
  q4  --eps--> {q6}          [correct]
  q5  --'b'--> {q6}          [correct]
  q6  --eps--> {q2, q7}     [correct]
  q7  --'a'--> {q8}          [correct]
  q8  --'b'--> {q9}          [correct]
  q9  --eps--> {q10, q21}   [WRONG: q21 belongs
    ↪  to (ba)* sub-NFA,
                                should be {q10}
    ↪ only at this point]
  q10 --eps--> {q11, q19}   [WRONG: q19
    ↪ incorrectly reachable here]
  ...
```

The No-CoT model, given the same question, produces the correct transition table directly without any intermediate steps. Language equivalence evaluation confirms the CoT model's output is incorrect (it accepts strings not in $L(((a|b)^*ab(ba)^*)+)$) while the No-CoT model's output is correct.

Interpretation. This failure pattern is consistent with template mimicry: the CoT model has learned to produce five well-structured steps that *look* like

Thompson's construction, but at high complexity it loses track of which states belong to which sub-NFA. The template provides a false sense of correctness – all steps are present and formatted properly, masking the underlying structural error. The No-CoT model, lacking any template to mimic, must compress the regex-to-NFA mapping directly into its weights, which appears to produce more reliable structural reasoning at high complexity.

D.2 No-CoT Fails on GSM8K: Multi-Step Dependencies

On GSM8K, No-CoT SFT collapses to 17.5% accuracy, well below even the zero-shot baseline of 71.0%. The following examples are drawn directly from confirmed evaluation logs and illustrate the failure mode: the No-CoT model produces numerically plausible single-step answers but fails on problems requiring intermediate results to be carried forward across multiple steps.

Example 1: Partial computation (index 194).

> **Question:** Toby is reading a book that is 45 pages long. It averages 200 words a page. Toby can read at a rate of 300 words per minute. He has to be at the airport in 60 minutes and plans to leave as soon as he finishes. How many minutes does Toby have left to read?

Ground truth: 20
No-CoT model output: FINAL ANSWER: 15
Analysis: The correct solution requires three chained steps: total words = $45 \times 200 = 9{,}000$; reading time = $9{,}000 \div 300 = 30$ minutes; remaining time = $60 - 30 = 20$ minutes. The model's answer of 15 suggests it computed one of the intermediate quantities correctly (possibly $60-45 = 15$, treating pages as minutes) but failed to chain the operations. Without a scratchpad, intermediate results cannot be accumulated.

Example 2: Missed accumulation step (index 196).

> **Question:** While at the dollar store, Sloane counts 100 customers entering the store. The next day, she counts 50 more customers than the first day. If the total number of customers by the third day was 500, how many customers entered on the third day?

Ground truth: 250
No-CoT model output: FINAL ANSWER: 200
Analysis: The correct solution is: Day 2 = $100 + 50 = 150$; Day 3 = $500 - 100 - 150 = 250$. The model outputs 200, which suggests it computed $500 - 100 - 200 = 200$, treating Day 2 as 200 (possibly 100×2) rather than 150. The error originates in the Day 2 computation and propagates directly to the final answer. With no intermediate scratchpad, the model cannot verify or correct the Day 2 result before using it.

Example 3: Correct answer (index 195).

> **Question:** A phone tree is used to contact families and relatives of Ali's deceased coworker. Ali decided to call 3 families. Then each family calls 3 other families, and so on. How many families will be notified after the 4th round of calls?

Ground truth: 81
No-CoT model output: FINAL ANSWER: 81
Analysis: The No-CoT model correctly answers this problem because it reduces to a single computation: $3^4 = 81$. No intermediate state needs to be carried forward – the answer is a direct function of the inputs. This confirms the pattern: No-CoT models can solve problems that reduce to a single operation but fail on problems requiring sequential accumulation of intermediate results.

Summary. The GSM8K failure examples reveal that No-CoT performance collapse is not a formatting issue – the model produces syntactically valid FINAL ANSWER: responses in every case. The failure is computational: without a scratchpad in which to accumulate intermediate results, the model can only perform effectively on single-step problems. CoT training, by contrast, provides explicit intermediate steps as training targets, teaching the model to break problems into sequential sub-computations. This is precisely the regime where answer backtrackability is zero: given only the integer 20, there is no way to recover the three-step chain $9{,}000 \div 300$, $60 - 30$ that produces it.

D.3 Tier 5 Failure Analysis for CoT on NFA

At Tier 5 of the NFA dataset (32–50 states), CoT SFT achieves only 27.0% accuracy for the 7B model – the lowest accuracy of any model-tier combination in our experiments. This section analyses the specific failure modes at Tier 5 to understand why CoT supervision breaks down at high complexity.

Failure mode 1: State counter overflow. Thompson's construction for complex regexes produces NFAs with 32–50 states, each requiring a unique label. The CoT model, despite being trained on examples with up to 50 states at Tier 5, begins to reuse state labels or skip states in its output at high complexity. The following illustrative output shows a model generating Steps 1–4 correctly but then producing an incomplete transition table in Step 5:

CoT model – Step 5 output truncated at state q28 (illustrative)

Step 5: NFA Transition Table

State	Role	a	b	eps (ϵ)
q1	start	empty	empty	{q2, q21}
q2		empty	empty	{q3, q5}
⋮			...	
q27		{q28}	empty	empty
q28		empty	{q29}	empty

NFA: 38 states ({q1, . . . , q38}), start = q1, accept = q38

[output truncated – remaining 10 states missing]

The model correctly states in the summary line that the NFA has 38 states but only outputs 28 rows in the table. The missing states cause the transition table to be incomplete and therefore incorrect under structural equivalence evaluation, which requires every state's transitions to be specified.

Failure mode 2: Sub-NFA boundary confusion. For regexes with deeply nested closures, the CoT model correctly identifies the Thompson rules in Step 2 but makes systematic errors in Step 3 when assigning epsilon transitions between sub-NFAs. Specifically, the model confuses the entry and exit epsilon transitions for nested star operators, producing NFAs that accept a superset or subset of the correct language. This is the template mimicry failure described in Section D.1 but is more prevalent at Tier 5 due to the greater number of nested sub-NFAs.

Failure mode 3: Correct trace, wrong table. In a substantial fraction of Tier 5 CoT failures, Steps 1–4 are entirely correct – the parse tree, Thompson rules, transition listing, and eps-closure are all accurate – but the final transition table in Step 5 does not match the transition listing in Step 3. The model appears to regenerate the table from scratch rather than transcribing it from Step 3, introducing transcription errors. This suggests that the CoT template creates a disconnect between the reasoning steps and the final answer: the model treats the table as a separate generation task rather than a direct copy of information already computed.

No-CoT at Tier 5. The No-CoT model achieves 52.0% at Tier 5 – nearly double the CoT accuracy. No-CoT failures at Tier 5 are qualitatively different: the model produces a complete and well-formed transition table, but with systematic errors in epsilon transitions that suggest the model has learned approximate heuristics for Thompson's construction rather than the exact procedure. For example, the model may correctly place all symbol transitions but miss some epsilon transitions in the loop-back structure of a Kleene star over a complex sub-expression. These are errors of incomplete learning rather than template mimicry, and they scale more gracefully with complexity.

Implication. The Tier 5 analysis suggests that CoT supervision introduces two compounding failure modes at high complexity: sequence length pressure (the model cannot fit all 50 states into a single forward pass without truncation or repetition) and template rigidity (the fixed five-step structure does not adapt to regexes that require more careful state bookkeeping). No-CoT supervision avoids both failure modes by not requiring any intermediate steps, at the cost of learning approximate rather than exact procedures – but approximation errors at Tier 5 are less damaging than template rigidity errors.

E Training and Evaluation Details

E.1 Hyperparameter Table

Table 12 reports the full hyperparameter configuration used for all LoRA fine-tuning experiments. All hyperparameters are held constant across CoT and No-CoT conditions, across all three model families, and across all five datasets. The only variables across runs are the training data file and the output directory. This design ensures that any performance difference between CoT and No-CoT conditions is attributable solely to the presence or absence of intermediate reasoning traces in the training data, and not to any difference in optimisation procedure, batch composition, or sequence length budget.

Table 13 reports confirmed training times for each dataset and model combination from SLURM job logs. CoT runs are consistently longer than No-CoT runs due to the longer target sequences in

CoT training examples.

Table 12: LoRA fine-tuning hyperparameters. All values are identical across CoT and No-CoT conditions and all models.

Hyperparameter	Value
Model loading	
Precision	bfloat16
Device map	auto (single GPU)
Gradient checkpointing	enabled
LoRA configuration	
Rank r	32
Alpha α	64
Dropout	0.05
Bias	none
Target modules	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
DoRA	disabled
Training	
Epochs (all datasets)	2
Per-device batch size	1
Gradient accumulation steps	8
Effective batch size	8
Learning rate	2×10^{-4}
LR scheduler	cosine decay
Warmup steps	50 (linear)
Weight decay	0.01
Optimizer	AdamW (fused)
TF32	enabled
Sequence length	
Maximum sequence length (NFA CoT)	8192 tokens
Maximum sequence length (NFA No-CoT)	8192 tokens
Maximum sequence length (DFA CoT)	8192 tokens
Maximum sequence length (DFA No-CoT)	8192 tokens
Maximum sequence length (math datasets)	4096 tokens
Checkpointing	
Eval strategy	every 200 steps
Save strategy	every 200 steps
Checkpoints retained	3 most recent
Hardware	
GPU	NVIDIA H100 NVL
GPU count	1
Training framework	HuggingFace Transformers + TRL
TRL version	1.3.0
Transformers version	5.8.0
PEFT version	0.18.1
Random seed	42 (all runs)

E.2 System Prompts and Evaluation Prompts

NFA dataset – training system prompt (CoT and No-CoT):

> *You are an expert in formal language theory and automata. When given a regular expression, you construct a Nondeterministic Finite Automaton (NFA) using Thompson's construction. You present the NFA as a clear transition table showing all states, symbol transitions, and epsilon transitions.*

Table 13: Training time per run on a single H100 NVL. CoT runs are longer due to longer target sequences.

Dataset	Model	CoT (hours)	No-CoT (hours)
NFA	1.5B	3.0	2.0
	7B	4.45	3.18
	8B	4.5	3.2
DFA	1.5B	3.19	2.97
	7B	4.17	3.02
	8B	4.41	3.35
GSM8K	1.5B	1.0	0.5
	7B	2.0	0.95
	8B	2.0	1.0
MATH	1.5B	1.5	0.8
	7B	2.5	1.2
	8B	2.5	1.2
MathInstruct	1.5B	4.0	2.0
	7B	8.0	4.0
	8B	8.0	4.0

NFA dataset – training user prompt (one of five templates):

> Convert the regular expression [regex] over the alphabet $\Sigma = \{[\text{alpha}]\}$ to an NFA using Thompson's construction. Show the resulting NFA transition table.

The remaining four training templates follow the same pattern with varied phrasing. All five explicitly name Thompson's construction and use backtick-formatted regex. The full set of OOD templates used for out-of-distribution evaluation is given in Appendix C.3.

DFA dataset – CoT training system prompt:

> *You are a DFA construction assistant. You must output the TRACE and FINAL DFA exactly in the same format as training data. Do not output any explanation outside the TRACE markers.*

DFA dataset – No-CoT training system prompt:

> *You are a DFA construction assistant. You must output ONLY the DFA in the exact required format. Do not provide any explanation, extra text, or commentary.*

DFA dataset – training user prompt:

> You are given a regular expression over alphabet $\Sigma = \{[\text{alpha}]\}$. Construct a complete minimised deterministic finite automaton (DFA) that recognizes exactly the same language as the given regular expression. [Show all steps / Output the DFA only.]
>
> REGEX: [regex]

Mathematical datasets – training system prompt (GSM8K, MATH, MathInstruct):

> *You are a helpful math assistant. Solve the problem step by step, then give the final answer clearly at the end. End your response with: FINAL ANSWER: ⟨your answer⟩*

Mathematical datasets – CoT training format: The full reasoning trace from the dataset is used as the assistant response, ending with FINAL ANSWER: [answer]. For GSM8K, the reasoning steps are taken directly from the `answer` field (everything before `####`), with the integer after `####` used as the final answer. For MATH, the solution field provides the reasoning and the answer field provides the final answer. For MathInstruct, the existing chain-of-thought rationale in the dataset is used directly.

Mathematical datasets – No-CoT training format: The assistant response contains only: FINAL ANSWER: [answer] with no reasoning steps.

Evaluation prompt (all datasets): At evaluation time, the system prompt is identical to the training system prompt. The user prompt is drawn from the test set. The assistant turn is left empty and the model generates its response greedily (do_sample=False, temperature=1.0, top_p=1.0). Maximum new tokens: 2048 for CoT models on formal-language datasets, 1024 for No-CoT models on formal-language datasets, and 512 for all models on mathematical datasets.

E.3 Structural Equivalence Evaluation

Correctness of NFA and DFA predictions is evaluated via *structural equivalence* of minimised automata. Both the predicted automaton and the ground truth are minimised using Hopcroft's algorithm, and a prediction is marked correct if and only if the two minimised automata are isomorphic up to state relabelling. This gives an exact decision over the full infinite language with no dependence on string length bounds.

Parsing predicted automata. The model output is parsed by locating the last markdown table containing both a State column and a Role column (for NFA) or the last FINAL DFA BEGIN. . .FINAL DFA END block (for DFA). If no valid automaton can be parsed from the output, the example is marked as a parse failure and counted as incorrect. Parse failure rates are reported separately from structural equivalence failures in all result tables.

State relabelling robustness. The structural equivalence metric is fully robust to state relabelling by construction: two minimised automata are compared by checking graph isomorphism up to state label permutation. A predicted DFA that uses state labels s1, s2, . . . instead of q1, q2, . . . receives full credit if the minimised transition structures are identical. This is important because the model is not explicitly constrained to use specific state names, and penalising correct automata for using different labels would produce misleadingly low accuracy scores.

Why structural equivalence rather than string simulation. An earlier version of our evaluation used language equivalence via string simulation up to a maximum length. We moved to structural equivalence for two reasons. First, string simulation up to any finite bound cannot guarantee correctness over the full infinite language – a predicted automaton could agree with the ground truth on all strings up to length L but diverge on longer strings. Second, structural equivalence via minimisation is computationally cheaper for large automata and gives an exact decision. The Myhill-Nerode theorem guarantees that the minimal DFA for a regular language is unique up to isomorphism, so structural comparison of minimised automata is a complete correctness criterion.

F Code and Reproducibility

All code, datasets, and model checkpoints are released publicly at our anonymous repository. This appendix describes the key components of the codebase and provides a step-by-step reproduction guide.

F.1 Repository Structure

Directory layout (confirmed from repository)

```
cot-vs-nocot/
|
|-- data_generation/
|   |-- nfa_parallel_dataset_gen_v2.py   # NFA
    ↪ dataset generator
|   `-- dfa_parallel_dataset_gen.py      # DFA
    ↪ dataset generator
|
|-- datasets/
|   |-- nfa/
|   |   |-- nfa_v2_cot_{train,val,test,ood_test,
    ↪ full}.jsonl
|   |   `-- nfa_v2_nocot_{train,val,test,
    ↪ ood_test,full}.jsonl
|   `-- dfa/
|       |-- dfa_cot_{train,val,test,ood_test,
    ↪ full}.jsonl
|       `-- dfa_nocot_{train,val,test,ood_test,
    ↪ full}.jsonl
|
|-- training/
|   |-- nfa/
|   |   |-- finetune_nfa_v2_cot.py
|   |   `-- finetune_nfa_v2_nocot.py
|   |-- dfa/
|   |   |-- finetune_dfa_cot.py
|   |   `-- finetune_dfa_nocot.py
|   `-- math/
|       |-- finetune_gsm8k_{cot,nocot}.py
|       |-- finetune_math_{cot,nocot}.py
|       `-- finetune_mathinstruct_{cot,nocot}.
    ↪ py
|
|-- evaluation/
|   |-- evaluate_nfa_v2.py
|   |-- evaluate_dfa.py
|   |-- eval_gsm8K.py
|   |-- eval_math.py
|   `-- eval_mathinstruct.py
|
|-- slurm/
|   |-- nfa/{submit_nfa_cot_v2.sh,
    ↪ submit_nfa_nocot_v2.sh}
|   |-- dfa/{submit_dfa_cot.sh,
    ↪ submit_dfa_nocot.sh}
|   `-- math/{submit_gsm8k.sh, submit_math.sh,
    ↪ run_mathinstruct.sh}
|
|-- analysis/
|   |-- compare_runs.py        # plots CoT vs
    ↪ No-CoT loss curves
|   `-- summarise_results.py   # aggregates
    ↪ eval JSONs into tables
|
|-- requirements.txt
`-- LICENSE
```

F.2 Dataset Generation Code

NFA dataset: `nfa_parallel_dataset_gen_v2.py`

Generates parallel CoT/No-CoT NFA datasets from the same regex instances, producing twelve output files in a single run. The key design decision is storing all examples as 4-tuples `(cot_iid, nocot_iid, cot_ood, nocot_ood)` throughout the pipeline and splitting only at write time, guaranteeing that CoT and No-CoT files are perfectly aligned at every index. The core functions are:

- `random_regex(tier, alphabet, rng, depth)`: generates a random regex AST using tier-specific operator weights and recursion depth limits.
- `thompson(node)`: converts a desugared AST to an NFA via Thompson's construction, returning an `NFA` object with a single start and accept state.
- `_build_parallel_entry(...)`: calls `thompson`, verifies correctness, draws one question template from each of `USER_TEMPLATES` and `OOD_USER_TEMPLATES`, builds both CoT and No-CoT responses, and returns the 4-tuple.
- `build_nfa_table(nfa, alphabet)`: renders the NFA as a markdown transition table. Called by both `build_cot_response` and `build_nocot_response`, guaranteeing the answer is byte-for-byte identical in both training conditions.

DFA dataset: `dfa_parallel_dataset_gen.py`

Generates parallel CoT/No-CoT DFA datasets. The three-phase construction pipeline (Thompson $\rightarrow$ subset construction $\rightarrow$ Hopcroft minimisation) is implemented in pure Python with no external automata libraries. A virtual dead state is introduced during Hopcroft minimisation and then removed from the final DFA. Verification cross-checks all three automata ($L(\text{NFA}) = L(\text{DFA}) = L(\text{minDFA})$) on all strings up to length 7.

F.3 Fine-Tuning Code

All training scripts share an identical structure and hyperparameters. The only differences across CoT and No-CoT conditions are the input data file and the output directory, ensuring that any performance difference is attributable solely to the training data. All scripts use `SFTConfig` (TRL 1.3.0), pass the tokenizer via `processing_class=tokenizer`, and include a `MetricsLogger` callback that writes training loss, eval loss, learning rate, and gradient norm to `training_metrics.csv` at every logging step.

Target modules confirmed from all training scripts: `q_proj`, `k_proj`, `v_proj`, `o_proj`,

gate_proj, up_proj, down_proj.

Note: MathInstruct scripts use warmup_steps=100 rather than 50, as the larger dataset size (257,000 training examples) warrants a longer warmup.

F.4 Evaluation Code

NFA and DFA evaluators: evaluate_nfa_v2.py, evaluate_dfa.py

Both evaluators accept -lora_path to load a LoRA adapter (merged before evaluation) or omit it for baseline evaluation. Correctness is evaluated via *structural equivalence* of minimised automata: both the predicted automaton and the ground truth are minimised using Hopcroft's algorithm, and a prediction is correct if and only if the two minimised automata are isomorphic up to state relabelling. This gives an exact decision over the full infinite language with no dependence on string length bounds. Parse failures – model outputs containing no valid automaton table – are counted separately from structural equivalence failures and reported in all result tables.

Math evaluators. eval_gsm8K.py covers GSM8K and MATH-500. eval_math.py covers Hendrycks MATH with per-subject and per-level breakdowns. eval_mathinstruct.py covers MathInstruct with per-source and CoT/PoT type breakdowns. All three use a two-stage answer extraction: first search for FINAL ANSWER: marker, then fall back to the last number in the output. Numeric answers are compared with tolerance 10^{-6}.

F.5 Step-by-Step Reproduction

Step 1. Install dependencies.

Environment setup

```
pip install -r requirements.txt
# torch>=2.1.0, transformers==5.8.0,
    ↪ trl==1.3.0,
# peft==0.18.1, datasets>=2.18.0,
    ↪ accelerate>=0.27.0
```

Step 2. Generate datasets. Datasets are already included in the repository under datasets/. To regenerate from scratch:

Dataset generation (30-60 minutes on CPU)

```
python3 data_generation/
    ↪ nfa_parallel_dataset_gen_v2.py \
    --n 25000 --seed 42 --out_prefix
    ↪ nfa_v2 --test_per_tier 200

python3 data_generation/
    ↪ dfa_parallel_dataset_gen.py \
    --n 25000 --seed 42 --out_prefix
    ↪ dfa --test_per_tier 200
```

Step 3. Fine-tune models. Edit MODEL_PATH at the top of each script, then submit SLURM jobs. CoT and No-CoT jobs can run in parallel – they coordinate via a sentinel file to avoid double dataset generation:

Submitting fine-tuning jobs

```
sbatch slurm/nfa/submit_nfa_cot_v2.sh
sbatch slurm/nfa/submit_nfa_nocot_v2.sh
sbatch slurm/dfa/submit_dfa_cot.sh
sbatch slurm/dfa/submit_dfa_nocot.sh
sbatch slurm/math/submit_gsm8k.sh
sbatch slurm/math/submit_math.sh
sbatch slurm/math/run_mathinstruct.sh
```

Step 4. Evaluate. Each SLURM script runs evaluation automatically after training. To evaluate a checkpoint manually:

Manual evaluation (NFA example)

```
python3 evaluation/evaluate_nfa_v2.py \
    --model_path /path/to/model \
    --lora_path  lora_output_nfa_v2_cot
    ↪  \
    --data_file  datasets/nfa/
    ↪ nfa_v2_cot_test.jsonl \
    --mode       cot \
    --out        eval_results/
    ↪ nfa_cot_iid.json
```

Step 5. Aggregate results.

Summarising all results

```
python3 analysis/summarise_results.py
    ↪ --results_dir eval_results/
```

Step 6. Plot loss curves. The MetricsLogger callback writes training_metrics.csv to each output directory during training. To plot CoT vs No-CoT curves:

Plotting training curves

```
python3 analysis/compare_runs.py \
    --cot_dir   lora_output_nfa_v2_cot
    ↪ \
    --nocot_dir
    ↪ lora_output_nfa_v2_nocot \
    --title     "NFA Dataset (7B)" \
    --out       figures/nfa_loss_curves.
    ↪ png
```

Hardware requirements. All experiments require a single NVIDIA H100 NVL GPU (80 GB VRAM). Peak memory usage is approximately 35 GB for the 7B model, 18 GB for LLaMA-8B, and 8 GB for the 1.5B model, all with LoRA and gradient checkpointing enabled. Minimum

32 GB system RAM is recommended. Total compute across all experiments is estimated at 150–200 GPU hours.

F.6 Dataset Repository

Both datasets are included directly in the repository under `datasets/nfa/` and `datasets/dfa/`:

Table 14: Dataset files in repository.

Dataset	Files	Size
NFA (CoT + No-CoT)	`datasets/nfa/nfa_v2_{cot,nocot}_*.jsonl`	25k pairs
DFA (CoT + No-CoT)	`datasets/dfa/dfa_{cot,nocot}_*.jsonl`	22.5k pairs