

The Imitation Game: Turing Machine Imitator is Length Generalizable Reasoner

Zhouqi Hua^{1,2*} Wenwei Zhang^{1*†} Chengqi Lyu¹ Yuzhe Gu^{1,3}
 Songyang Gao¹ Kuikun Liu¹ Kai Chen^{1†}
¹Shanghai AI Laboratory ²Fudan University ³Shanghai Jiao Tong University
 {huazhouqi,zhangwenwei,chenkai}@pjlab.org.cn

Abstract

Length generalization, the ability to solve problems of longer sequences than those observed during training, poses a core challenge of Transformer-based large language models (LLM). Although existing studies have predominantly focused on data-driven approaches for arithmetic operations and symbolic manipulation tasks, these approaches tend to be task-specific with limited overall performance. To pursue a more general solution, this paper focuses on a broader case of reasoning problems that are *computable*, *i.e.*, problems that algorithms can solve, thus can be solved by the Turing Machine. From this perspective, this paper proposes **Turing Machine Imitation Learning (TAIL)** to improve the length generalization ability of LLMs. TAIL synthesizes chain-of-thoughts (CoT) data that imitate the execution process of a Turing Machine by computer programs, which *linearly* expands the reasoning steps into *atomic* states to alleviate shortcut learning and explicit *memory* fetch mechanism to reduce the difficulties of dynamic and long-range data access in elementary operations. To validate the reliability and universality of TAIL, we construct a challenging synthetic dataset covering 8 classes of algorithms and 18 tasks. Without bells and whistles, TAIL significantly improves the length generalization ability as well as the performance of Qwen2.5-7B on various tasks using only synthetic data, surpassing previous methods and DeepSeek-R1. The experimental results reveal that the key concepts in the Turing Machine, instead of the thinking styles, are indispensable for TAIL for length generalization, through which the model exhibits read-and-write behaviors consistent with the properties of the Turing Machine in their attention layers. This work provides a promising direction for future research in the learning of LLM reasoning from synthetic data.

1 Introduction

Length generalization [1], *i.e.*, the ability to handle a problem with input sequences of various lengths in the open world, especially those longer than previously seen, is a fundamental aspect of human intelligence and serves as a crucial evaluation criterion for AI systems [2–5]. Although the ability and generalizability of large language models (LLMs) to solve complex problems have been significantly improved by chain-of-thought (CoT) techniques [6], recent studies [2, 7, 8] indicate that LLMs still struggle with length generalization, which sometimes explores and falls into shortcuts that eventually cause errors [9].

To address the challenge, existing works [8, 10–13] primarily focus on data-driven approaches, which refine the training data of LLMs by modifying the structure of CoT data to be more effective and generalizable. However, these methods remain inherently task-specific (*e.g.*, Index Hint [8, 10] for symbolic reasoning tasks and Reversed Format [10–13] for arithmetic problems) and yield only

† Corresponding author

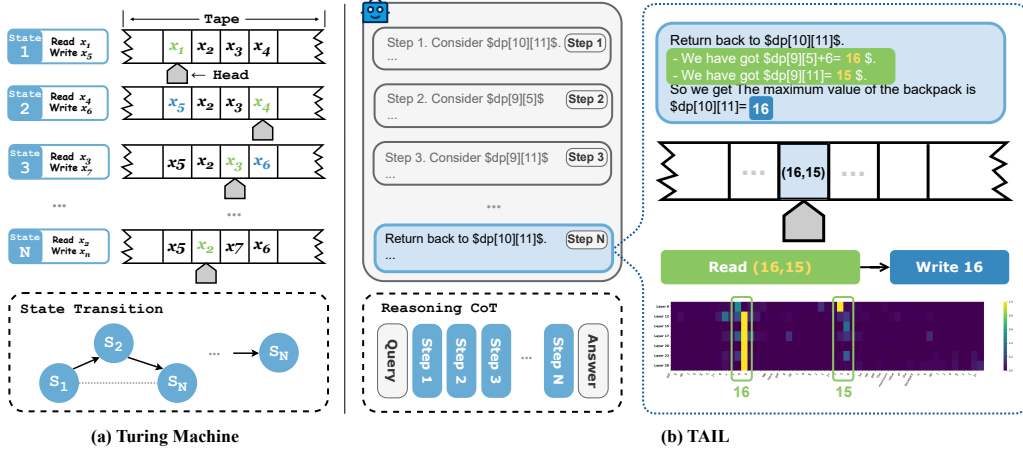


Figure 1: Construction process of TAIL. (a) Schematic diagram of a Turing machine executing a program. (b) The basic principle of TAIL. Visualization of attention in the Transformer layers shows that the model has learned to imitate the behavior of a Turing machine.

moderate performance. So a question arises: *Is there a universal and effective reasoning structure of CoT for length generalization?*

This paper aims to answer this question by first taking a deeper look at the common property of the problems. Notably, we observe that many of these tasks admit well-defined stepwise procedures that can be solved by program algorithms that generalize to inputs of arbitrary length, which are also referred as **Computable Problems**. Thus, the core of achieving length generalization lies in letting LLMs completely simulate the execution process of the corresponding programs within their CoT for each problem. In essence, the LLM acts like a Turing Machine (Figure 1(a)), performing a sequence of fundamental operations on a memory tape, guided by finite states and logical control.

From this perspective, we propose **Turing Machine Imitation Learning (TAIL)**, which contains the three key structures in the synthesized CoT data that emulate three core properties of Turing machine execution: **Linear Transition**, **Atomic State**, and **Memory Fetcher**. First, similar to the Turing Machine execution process, **Linear Transition** enforces a complete and linear arrangement of reasoning steps to eliminate potential shortcut learning. Second, TAIL decomposes the reasoning content into minimal units, termed **Atomic States** to reduce difficulty and further reduce shortcut learning, which essentially correspond to the states of a Turing Machine, including read, write, and logical control operations. Third, because LLMs can only append instead of modify in-place the tokens in their context due to their auto-regressive nature, the context of LLMs, which essentially serves as a memory, will keep growing as the reasoning continues. This poses difficulties for LLMs because of their attention mechanisms when they need to conduct elementary operations on operands that have long and dynamic distances among them. Therefore, TAIL further adopts a mechanism, termed **Memory Fetcher**, to read the necessary operand data and explicitly output them in the current step before conducting elementary operations.

To assess the universality and effectiveness of TAIL, we construct a challenging dataset spanning 18 tasks across 8 algorithms—substantially harder than those in prior length generalization studies. Fine-tuning Qwen2.5-7B [14] on this dataset yields high label accuracy across length ranges, with consistent gains on longer sequences, demonstrating strong length generalization. The model outperforms prior methods and surpasses DeepSeek-R1 [15]. Ablation studies show that removing any core module of TAIL severely degrades long-sequence performance. Notably, even minimalist CoT data containing only core modules without any thinking styles maintains full effectiveness, confirming TAIL as the key data-driven enabler of length generalization. We also visualize the attention maps of the TAIL-fine-tuned model and observe that the attention during write operations focuses on fetched operands within the same state, resembling Turing machine behavior (Figure 1(b)).

2 Related Work

Length Generalization. Large language models (LLMs) exhibit significant limitations in length generalization, often struggling to process input sequences longer than those encountered during

training [2, 7, 16, 17]. Previous works primarily focus on model architecture enhancements and data-driven approaches to improve the length generalization. However, architectural enhancements modify the components (*e.g.*, forward mechanism [18], attention mechanisms [19], and position encodings [20–22]) of Transformers for specific tasks, which have not apply to LLMs that have been pre-trained on large-scale data. Data-driven approaches construct special chain-of-thought (CoT) structures for training, such as digit-order reversal [8, 10–13, 23], sequence padding [24], and index hints [8, 10], which are task-specific and lack universality. Similar to ours, recent work [25] uses a Turing-like approach to decompose the task into step-by-step tape updates, but it only works under specific positional encoding and data combinations and has not been verified to be universal on a wide range of tasks. Our TAIL focuses on universal data-driven approaches, exploring a more general and effective CoT structure, and directly adopting mainstream LLMs [14, 26–30] for fine-tuning, without modifying any components of the pre-trained model.

Structured Chain-of-Thought Construction. Structured thinking demonstrably enhances the reasoning capabilities of LLMs [6]. Building upon this foundation, prior research has explored various recognition heuristics within the Chain-of-Thought (CoT) paradigm, aiming to imbue LLMs with more human-like thought processes [31–33]. Concurrently, investigations into diverse structured data formats, including linear chains [6], hierarchical trees [34], interconnected graphs [35], and dynamically adapting structures [36], which enable LLMs to search easily and improve the complex problem-solving performance. In this paper, we introduce a novel approach to synthesizing structured CoT data by drawing inspiration from the architecture of a Turing Machine. This emulation offers a theoretically powerful advantage: length-generalizability, enabling the model to tackle problems of varying complexity, and broad applicability to the entire domain of *computable* problems. Notably, the search capability and different graph structures [34–36] and their targeted tasks can all be taken as instances of a Turing Machine solving computable problems.

Eliciting Chain-of-Thought. Efforts to guide LLMs in generating CoT rationales with specific, desired structures primarily involve prompting, supervised fine-tuning (SFT), and reinforcement learning (RL). Prompting-based methods, encompassing static example-based approaches [37–39] and adaptive generation strategies [40–42], guide the model through carefully crafted prompts. However, these methods do not fundamentally enhance the inherent reasoning capabilities of LLMs. SFT methods [43–46] emphasize guiding a single LLM or a combination of multiple LLMs to synthesize training data within specific CoT structures, enabling the model to learn and imitate the desired reasoning behavior. RL-based methods [47–49] allow the model to generate its own CoT reasoning traces and optimize the LLM directly through a reward function, whose resulting CoT structures emerge during training and thus are not fully controllable. Despite the progress of SFT and RL, a persistent challenge remains to mitigate shortcuts and logical omissions during CoT reasoning, especially when relying on model-generated rollouts [43, 46, 50, 51]. To address this, we explore synthesizing CoT data for SFT using code programs to guarantee that the structure of CoT strictly mimics the execution process of a Turing Machine to reduce the possibility of learning shortcuts.

3 Preliminaries

3.1 Length Generalization and Computable Problems

For large language models (LLMs), length generalization means that a model can process long input sequences, although it is only trained on short sequences. For example, a model trained on 10–30 digit addition can maintain strong performance on 30–50 digit addition tasks. Fundamentally, successful length generalization implies that the model has extracted a structural pattern from the training data. This pattern should be general and can scale adaptively with input length.

After a deeper look at the problem, we observe that many tasks can essentially be solved through discrete symbolic transformations governed by bounded algorithmic computational rules [52–55]. For example, Parity can be solved through a simple enumeration procedure, while arithmetic addition can be handled by simulating the full digit-wise addition process, including carry propagation. We refer to such tasks as *Computable Problems*, whose commonality lies in being solvable by a well-defined, deterministic algorithmic procedure. The inherent ability of such algorithms to handle inputs of arbitrary length aligns naturally with the goal of length generalization, meaning that training LLMs to learn the step-by-step execution of such algorithms can enable them to generalize to different input lengths when solving computable problems.

3.2 Turing Machine

While all computable problems are solvable by algorithms, their structural diversity makes task-specific chain-of-thought (CoT) design impractical. Therefore, a more abstract and general framework is essential to unify the CoT paradigm for computable problems. Based on the **Church-Turing thesis** [56], a Turing Machine can solve any algorithmically computable problem, thereby providing a universal and higher-level framework for problem solving. In other words, the computational trace data of any computable problem can be constructed by simulating the execution of a Turing Machine.

The formal definition of the Turing Machine [52, 57] consists of an infinite-length *tape*, a read/write *head*, and a *table* containing a finite set of state transitions. It can be represented as a 7-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F), \quad (1)$$

where Q is a finite set of states, Σ is a non-empty finite input alphabet, Γ denotes the set of tape symbols, δ is the transition function, $q_0 \in Q$ refers to the initial state, $B \in \Gamma - \Sigma$ is the blank symbol, and F denotes the set of final states. In any non-accepting state q_s , the head reads a symbol a from the tape, overwrites it with a new symbol b , and moves the head to a new position, thereby transitioning to the next state q_{s+1} , which can be formally defined as:

$$\delta(q_s, a) = (q_{s+1}, b, D), \quad (2)$$

where the head moves one position in direction D . Thus, δ represents a complete state transition conflating two logically independent states, and a linear unfolding of states $q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_n$ represents the complete process of Turing machine implementing the program.

In order to align the reasoning process of LLMs with Turing Machine, the reasoning procedure can be unfolded into multi-step reasoning with the help of CoT, which can be represented as:

$$P_\theta(x) = \prod_{i=0}^n P(x_i | x_{<i}, \theta), \quad (3)$$

where n represents the total steps of reasoning. Each single reasoning step can be formalized as x_i , deriving the current reasoning result based on the preceding reasoning steps $x_{<i}$. It is important to note that the granularity of x_i is determined by the size of the reasoning step in the specific task. Typically, it corresponds to the prediction of multiple tokens, forming an intermediate reasoning outcome at each stage. x_0 represents the query and thus the complete reasoning path can be expressed as $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n$.

In line with the Turing Machine, each single-step reasoning step x in E.q.(3) corresponds to a Turing Machine state q in E.q.(2), which includes reading an input symbol a . The entire CoT is formed by a linear composition of such steps, analogous to the full unrolling of the Turing Machine’s state transitions δ from q_1 to q_n .

Previous work [58] has theoretically shown that Transformers can achieve Turing Completeness given sufficiently long chains of thought. However, it has not provided concrete guidelines for constructing such CoT sequences in a wide range of tasks. Based on our analysis, multi-step reasoning CoT can be structurally aligned with the computation process of a Turing machine. This leads us to hypothesize that, by endowing the reasoning process of LLMs with key properties of a Turing machine, the model can effectively simulate algorithmic execution and achieve length generalization.

4 Turing Machine Imitation Learning

Based on the preceding analysis, this paper proposes **Turing Machine Imitation Learning (TAIL)** to align the reasoning process of large language models (LLMs) simulate the execution of a Turing machine for achieving universal length generalization. TAIL aligns the structure of Chain-of-Thought (CoT) with the key properties of a Turing Machine (Figure 2) and comprises three core modules spanning from the macro-level to the micro-level: Linear Transition (Section 4.1), Atomic State (Section 4.2) and Memory Fetcher (Section 4.3).

4.1 Linear Transition

According to the RASP-Generalization Conjecture [10], Transformer-based LLMs struggle with problems that involve intricate control structures, such as loops. This suggests the need to transform

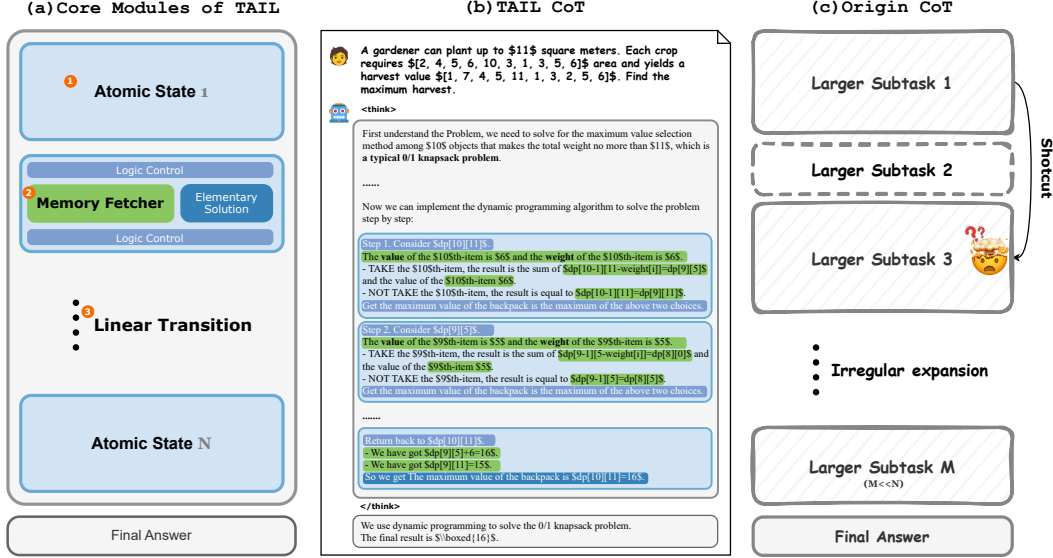


Figure 2: An overview of TAIL. (a) **Core Modules of TAIL** imitate a Turing Machine, containing a Linear Transition of Atomic State with Memory Fetcher of previous reasoning results. (b) **CoT generated by TAIL**: the solution to a 0/1 knapsack problem using a dynamic programming algorithm. (c) **Conventional CoT** consists of oversized subtasks, shortcut learning, and irregular expansion.

these structures into simpler forms that align better with the capabilities of the model. In particular, complex reasoning structures—like trees and graphs—can be linearly unrolled and traversed to enable complete and non-redundant execution of all reasoning steps, thereby preventing shortcuts in the reasoning process. Similarly, in a Turing machine, the execution of a complete program corresponds to a linear unfolding of states $q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_n$ as shown in Eq.(2), where even control structures such as loops can be flattened into a sequential process. To align with this characteristic, we introduce Linear Transition, which describes from a macro-level perspective how individual reasoning steps are composed into a linear and orderly structure within the overall reasoning process, and collectively form the CoT.

4.2 Atomic State

Although Linear Transition defines the overall structure of CoT reasoning as a linear sequence of reasoning steps, it does not impose constraints on the size of each step. Overly large reasoning steps not only increase the difficulty of learning for the model but also risk introducing shortcuts within a single step. Therefore, we attempt to constrain the size of a reasoning step by enforcing a standardized internal structure. Inspired by the Turing machine, each state encompasses a sequence of simple operations: *reading* data from the tape, *writing* new data, and *transitioning* to the next state. Following this principle, we define Atomic States consisting of operand retrieval (realized via Memory Fetcher, detailed in Section 4.3), the elementary solution produced within the reasoning step, and a set of logical control statements, as shown in Figure 2(a). Meanwhile, following the RASP-L hypothesis¹, we argue that each Atomic State should adhere to the principles of realizability and simplicity, and be decomposed into units that are as fine-grained as reasonably possible.

4.3 Memory Fetcher

In a Turing machine, every state reads data from the tape and writes new data at the same location. However, autoregressive models such as Transformers do not support in-place token modification. Instead, they extend the token sequence by appending new tokens, such as simulating result writing by appending the elementary solution, and guiding state transitions through explicit logic control.

¹We do not strictly follow RASP-L to constrain each reasoning step, but use it to indicate problems directly solvable by Transformers. This relaxed view shows strong length generalization in our experiments.

Moreover, data reading is typically achieved through the model’s attention mechanisms over previously generated tokens. As reasoning progresses, the sequence grows longer, requiring the model to attend over increasingly distant and dynamically shifting tokens. Furthermore, simultaneously performing data retrieval and generating the elementary solution increases the learning difficulty for the model. To address this, we propose Memory Fetcher, which decouples the act of establishing long-range attention from the actual execution of reasoning operations. By explicitly outputting the operands involved in the current reasoning step, the model can form more localized attention patterns, as illustrated in Figure F1. Without Memory Fetcher, the model tends to exhibit sparse attention over long token sequences, leading to insufficient focus on the actual operands.

5 Experiment

5.1 Dataset Synthesis

Task Selection. We synthesize a set of challenging tasks based on 8 classic algorithmic paradigms in computable problems (*i.e.*, Simulation, Recursion, Iteration, Greedy, Enumeration, DP, Divide & Conquer and Backtracking) to verify the effectiveness of TAIL. As shown in Table A1, the dataset comprises 18 tasks, including previously studied problems such as addition, but with randomized digit lengths to increase task difficulty.

Synthesis Approach. For each computable problem, we can easily construct an algorithm to solve it. As shown in Figure 2(b), we generate TAIL CoT data in bulk by: (1) treating each algorithmic step as an Atomic State, (2) unfolding the algorithmic process sequentially as Linear Transition, and (3) printing the input of the current algorithm step as Memory Fetcher within the CoT representation. During the generation process, we performed strict de-duplication and ensured that none of the data in the review set was included in the training set (those with the same operands but different query expressions were also eliminated). For training data, we first validated the sufficiency of the TAIL architecture by synthesizing *minimalist reasoning chains* that only include three core modules in a format similar to that shown in Figure B2 (see Appendix B). In addition, we also enrich the minimalist reasoning chains into *user-friendly CoT of TAIL* using natural language (see Figure B3) and verify length generalization ability on all 18 tasks, as shown in Figure 3.

Dataset Size. For most of the tasks, 100,000 training samples and 500 evaluation samples were included, and a few of the more difficultly constructed tasks contained 50,000 training samples and 200 evaluation samples. Each task has a high degree of diversity in query narratives, some of which incorporate real-world problems (*e.g.* Diophantine Equation, 0-1 Knapsack, etc). We divide the data for each task into three ranges of sequence lengths, *i.e.*, short (S), medium (M) and long (L). The length of each task is detailed in Table D1. The model is fine-tuned using data from the shortest range (S) and evaluated across all length ranges (S, M and L).

5.2 Experimental Settings

Metrics. Previous work [7] has demonstrated experimentally that *label accuracy* is well suited to measure reasoning capability of LLMs. We use pass@1 label accuracy under the zero-shot setting and use greedy decoding to evaluate.

Training. We fine-tuned Qwen2.5-7B across multiple tasks, with training epochs ranging from 2 to 5 for most tasks and exceeding 5 epochs for a few more challenging ones with a global batch size of 1024. The initial learning rate was $1e-5$, decaying to $7e-7$, with a weight decay of 0.1.

Evaluation. To facilitate a more efficient evaluation procedure, our evaluation follows a dual-model framework. First, a small 1.5B specialized model extracts the answers. Then, Qwen2.5-72B-Instruct performs evaluations, assessing correctness and outputting `\boxed{YES}` or `\boxed{NO}` to represent the evaluation result. Since our inference tasks have a single ground-truth answer, the evaluation model enforces strict content matching while allowing some flexibility in answer formatting.

5.3 Performance

Overall Performance. After fine-tuning on 18 tasks using user-friendly CoT of TAIL (as shown in Figure B3), we observed length generalization in multiple algorithms. Several tasks like *Compare Numbers*, *Bubble Sort* and *Any Substring* reach near saturation in both in-domain and out-of-domain

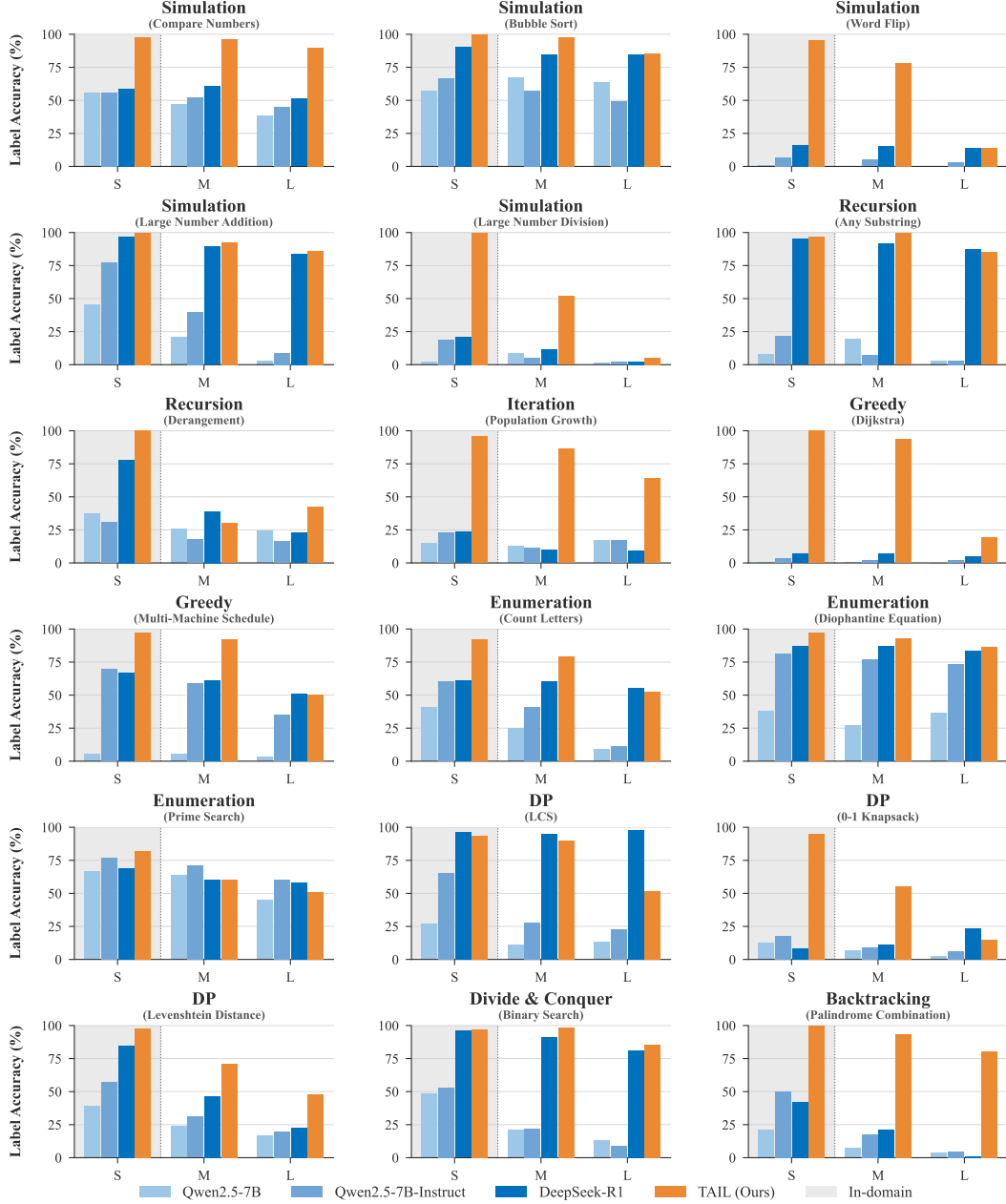


Figure 3: Length generalization across all 18 tasks in comparison with Qwen2.5-7B Instruct and DeepSeek-R1. It can be observed that the model demonstrates stronger generalization in reasoning over longer sequences, outperforming DeepSeek-R1 on the majority of tasks.

sequences. As shown in Figure 3, TAIL substantially enhances the long-sequence reasoning capabilities of the 7B model, enabling it to outperform DeepSeek-R1 on the majority of tasks. Compared to the significant performance degradation typically observed on longer sequences (with Qwen2.5-7B-Base as the baseline and Qwen2.5-7B-Instruct representing a conventionally CoT-trained counterpart), TAIL consistently maintains high label accuracy across most tasks, demonstrating strong generalization to longer contexts. Notably, even when approaching the maximum token limit, the model retains stable reasoning and achieves high accuracy, underscoring its robustness under extreme sequence lengths. Although evaluation was conducted on our benchmark dataset, the evaluation process allowed flexibility in formatting, ensuring a more objective assessment. We observed that when reasoning over long-sequence problems, DeepSeek-R1 often attempts multiple specific values in the phase and exploits shortcuts to bypass the structured reasoning process.

	S	M	L
Index Hint [8, 10–13, 23]	57.0	34.5	24.0
Reversed Format [8, 10]	39.5	35.5	35.0
TAIL (Ours)	97.0	92.5	86.5

Table 1: Pass@1 accuracy comparison of TAIL with previous works, *i.e.* index hint and reversed format, in large number addition task (belonging to simulation algorithm).

Model	Simulation		Enumeration		Iteration		Divide & Conquer		Recursion		DP		Greedy		Backtracking	
	M	L	M	L	M	L	M	L	M	L	M	L	M	L	M	L
Qwen2.7-7B Base	47.4	38.6	26.8	36.8	12.4	17.0	21.6	13.6	19.4	3.0	11.2	13.0	5.4	3.4	7.2	4.0
TAIL w/o Atomic State	82.6	73.6	68.2	69.0	77.2	61.2	86.2	71.0	52.2	32.0	77.4	61.0	39.0	16.0	75.8	61.4
TAIL w/o Linear Transition	80.0	75.4	63.6	58.8	76.2	54.0	85.0	75.6	43.0	30.8	77.4	74.2	20.6	11.2	79.0	62.8
TAIL w/o Memory Fetcher	90.2	88.0	64.2	63.6	73.8	67.6	92.4	88.2	87.2	84.8	80.8	74.8	45.6	30.8	80.0	69.2
TAIL	94.2	90.0	96.0	92.6	90.0	84.4	98.6	89.2	99.6	87.0	90.2	71.8	91.8	62	83.6	76

Table 2: Ablation study in core modules of TAIL. For each algorithm, we select a representative task and evaluate pass@1 accuracy only on sequences that exceed the training length. Clearly, the absence of any core module leads to a sharp degradation in length generalization performance.

Comparison with Previous Works. We compared TAIL with *Index Hint*[8, 10–13, 23] and *Reversed Format*[8, 10] on large-number addition, a task chosen for its difficulty and relevance to long-sequence generalization. Unlike prior work that used fixed-length integers, our setup samples two operands with random lengths and optional decimal points, greatly expanding the state space due to varied decimal placements. See Appendix F for detailed data construction methods. As shown in Table 1, using the same amount of data (100,000 training, 200 test), TAIL significantly outperforms both baselines on long sequences. The steep performance drop for prior methods, compared to their success on integer addition, highlights their task-specific nature and limited robustness to complexity.

Length Generalization Activation. For tasks that fail to achieve saturation on longer sequences, we gradually introduce longer examples into the training set. We then analyze the optimal proportion that enables effective length generalization at minimal training cost. Given that our dataset consists of three sequence length proportions, we explored five training configurations while keeping the total number of samples constant: <1:0:0> (the previous method using only short sequences), <8:1:1>, <7:2:1>, <5:3:2>, and <4:3:3>. As shown in Figure C1 (Appendix C), for most tasks, we observed that even a small addition of long-sequence data (*e.g.*, at an 8:1:1 ratio) led to a rapid saturation in long-sequence reasoning, a phenomenon we refer to as *length generalization activation*. Guided by this activation effect, CoT data synthesized using TAIL can leverage imbalanced sequence length proportions to reduce training costs.

5.4 Ablation Study

Impact of each core module of TAIL. To assess the necessity of each core module, we ablate them individually and examine the performance drop. As shown in Table 2, removing any module leads to a notable decline in length generalization. Importantly, the impact varies by task: for example, Data Review is critical for Population Growth (iteration-based), but less so for Compare Numbers (simulation-based). We attribute this variation to differences in task structure. Tasks like Compare Numbers involve only local transitions and weak long-range dependencies, making Data Review less essential. In contrast, recursion-heavy tasks benefit significantly from Linear Expansion. Most tasks are also sensitive to scale, highlighting the general need for Atomic Subtask decomposition. Overall, TAIL integrates all three modules synergistically to support diverse reasoning structures.

Impact of the CoT style. To investigate the influence of different CoT styles on performance, we conducted fine-tuning experiments using both standard CoT data and user-friendly CoT data. As illustrated in Figure 4, the results indicate that the choice of CoT style has minimal impact on the final performance. This suggests that for the length generalization task, the specific style of CoT is not a critical factor. Instead, the Turing-like mode appears to play a more significant role in determining the overall performance.

Attention visualization of Memory Fetcher. As shown in Figure F1, when Memory Fetcher is present, we observe strong and focused attention on the corresponding tokens (highlighted in selected

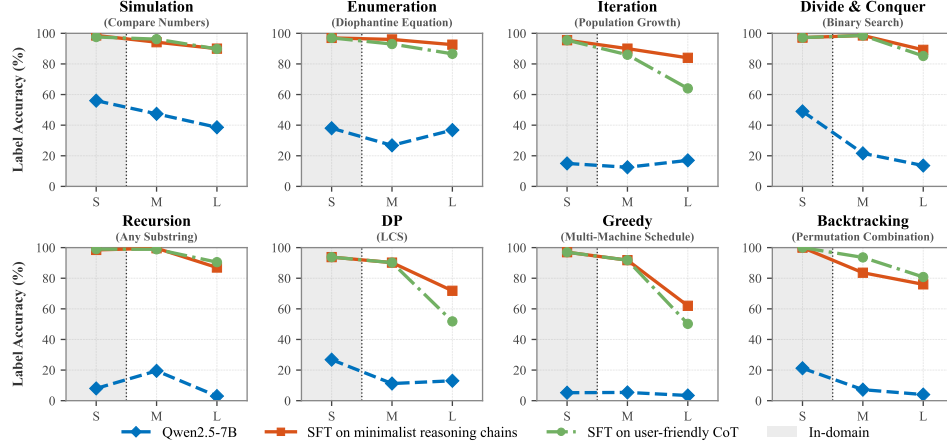


Figure 4: Comparison of fine-tuned Qwen2.5-7B with TAIL core module and the base model. For each algorithm, we select a representative task. After fine-tuning, model demonstrates length generalization on sequences that are 5 to 10 times longer than those in training.

Transformer layers). In contrast, without Memory Fetcher, the attention patterns become sparse and disorganized, showing insufficient focus on the operands.

6 Discussion

Task generalization problem. Although TAIL improves the length generalization performance on each single task, the training of one task does not significantly improve the performance of other tasks under the same algorithm (as shown in Appendix G). We believe that this is a gap between length generalization and combination generalization, which will also be our future work.

Inference length Limitation. Although TAIL significantly enhances the length generalization of large language models (LLMs), its core modules inevitably induce substantial CoT length expansion. Atomic State decomposes reasoning into fine-grained steps; Tape Review requires explicit output of all intermediate results before each subproblem, as these serve as operands for subsequent inference; and Linear Transition enforces a linear scaling between CoT length and task time complexity. Consequently, under constraints such as LLMs’ maximum token limits, inference latency, and memory usage, excessively long CoTs become a bottleneck for scaling to more complex problems. For instance, in *Permutation Combination* task, any algorithm that exhaustively generates all solutions must enumerate all $n!$ permutations, leading to a time complexity of $O(n!)$. Consequently, in such scenarios, the feasible problem size is restricted to the range $S = [1, 3]$, $M = [4, 6]$, $L = [7, 10]$.

7 Conclusion and Future Work

In this work, we propose Turing Machine Imitation Learning (TAIL), a data-driven framework for enhancing length generalization of large language models (LLMs). Instead of designing task-specific CoT structures for limited tasks, TAIL provides general design principles to synthesize CoT data for a broader range of problems, which are *computable*. TAIL formulate the process of solving computable problems in LLMs as a Turing machine execution process and introduce three core modules that facilitate length generalization: Linear Transition to avoid shortcut learning, Atomic State to reduce difficulty of each subtask, and Memory Fetcher to ease the access of dynamic and long-range data in memory for the operation of current steps. By fine-tuning Qwen2.5-7B on a challenging dataset spanning 8 algorithmic categories and 18 tasks, TAIL achieve significant length generalization, surpassing DeepSeek-R1 in long-sequence reasoning.

Our findings suggest that we have identified the key data modules that drive length generalization, supported by extensive validation and analysis. However, we also observe limited cross-task generalization, meaning that training on a single task within an algorithm provides only marginal benefits for other tasks, and cross-algorithm generalization remains weak. Addressing this challenge will be the focus of our future work.

References

- [1] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.
- [2] Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.
- [3] Sania Sinha, Tanawan Premisri, and Parisa Kordjamshidi. A survey on compositional learning of ai models: Theoretical and experimental practices. *arXiv preprint arXiv:2406.08787*, 2024.
- [4] Kartik Ahuja and Amin Mansouri. On provable length and compositional generalization. *arXiv preprint arXiv:2402.04875*, 2024.
- [5] Kensen Shi, Joey Hong, Manzil Zaheer, Pengcheng Yin, and Charles Sutton. Compositional generalization and decomposition in neural program synthesis. *arXiv preprint arXiv:2204.03758*, 2022.
- [6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [7] Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.
- [8] Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly. *arXiv preprint arXiv:2402.09371*, 2024.
- [9] Abulhair Saparov, Srushti Pawar, Shreyas Pimpalgaonkar, Nitish Joshi, Richard Yuanzhe Pang, Vishakh Padmakumar, Seyed Mehran Kazemi, Najoung Kim, and He He. Transformers struggle to learn to search. *arXiv preprint arXiv:2412.04703*, 2024.
- [10] Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.
- [11] Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*, 2023.
- [12] Ruoqi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. Positional description matters for transformers arithmetic. *arXiv preprint arXiv:2311.14737*, 2023.
- [13] Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, et al. Transformers can do arithmetic with the right embeddings. *Advances in Neural Information Processing Systems*, 37:108012–108041, 2024.
- [14] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [15] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [16] Yann Dubois, Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. Location attention for extrapolation to longer sequences. *arXiv preprint arXiv:1911.03872*, 2019.
- [17] Benjamin Newman, John Hewitt, Percy Liang, and Christopher D Manning. The eos decision and length extrapolation. *arXiv preprint arXiv:2010.07174*, 2020.

- [18] Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024.
- [19] Shaoxiong Duan, Yining Shi, and Wei Xu. From interpolation to extrapolation: Complete length generalization for arithmetic transformers. *arXiv preprint arXiv:2310.11984*, 2023.
- [20] Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bannani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. *arXiv preprint arXiv:2305.16843*, 2023.
- [21] Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. Functional interpolation for relative positions improves long context transformers. *arXiv preprint arXiv:2310.04418*, 2023.
- [22] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36:24892–24928, 2023.
- [23] Gonzalo Martínez, Lauren Watson, Pedro Reviriego, José Alberto Hernández, Marc Juárez, and Rik Sarkar. Combining generative artificial intelligence (ai) and the internet: Heading towards evolution or degradation? *arXiv preprint arXiv:2303.01255*, 2023.
- [24] Samy Jelassi, Stéphane d’Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*, 2023.
- [25] Kaiying Hou, David Brandfonbrener, Sham Kakade, Samy Jelassi, and Eran Malach. Universal length generalization with turing programs. *arXiv preprint arXiv:2407.03310*, 2024.
- [26] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- [27] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [28] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- [29] InternLM Team. Internlm: A multilingual language model with progressively enhanced capabilities, 2023.
- [30] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*, 2024.
- [31] Mirac Suzgun and Adam Tauman Kalai. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024.
- [32] Anni Zou, Zhuosheng Zhang, Hai Zhao, and Xiangru Tang. Generalizable chain-of-thought prompting in mixed-task scenarios with large language models. *arXiv preprint arXiv:2310.06692*, 2023.
- [33] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*, 2023.
- [34] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

- [35] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.
- [36] Tushar Pandey, Ara Ghukasyan, Oktay Goktas, and Santosh Kumar Radha. Adaptive graph of thoughts: Test-time adaptive reasoning unifying chain, tree, and graph structures. *arXiv preprint arXiv:2502.05078*, 2025.
- [37] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Synthetic prompting: Generating chain-of-thought demonstrations for large language models. In *International Conference on Machine Learning*, pages 30706–30775. PMLR, 2023.
- [38] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.
- [39] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.
- [40] KaShun Shum, Shizhe Diao, and Tong Zhang. Automatic prompt augmentation and selection with chain-of-thought from labeled data. *arXiv preprint arXiv:2302.12822*, 2023.
- [41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [42] Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2402.10200*, 2024.
- [43] Bin Yu, Hang Yuan, Yuliang Wei, Bailing Wang, Weizhen Qi, and Kai Chen. Long-short chain-of-thought mixture supervised fine-tuning eliciting efficient reasoning in large language models. *arXiv preprint arXiv:2505.03469*, 2025.
- [44] Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7601–7614, 2024.
- [45] Seungone Kim, Se June Joo, Doyoung Kim, Joel Jang, Seonghyeon Ye, Jamin Shin, and Minjoon Seo. The cot collection: Improving zero-shot and few-shot learning of language models via chain-of-thought fine-tuning. *arXiv preprint arXiv:2305.14045*, 2023.
- [46] Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. Chain of preference optimization: Improving chain-of-thought reasoning in llms. *Advances in Neural Information Processing Systems*, 37:333–356, 2024.
- [47] Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, et al. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-thought. *arXiv preprint arXiv:2501.04682*, 2025.
- [48] Yunhao Tang, Sid Wang, and Rémi Munos. Learning to chain-of-thought with jensen’s evidence lower bound. *arXiv preprint arXiv:2503.19618*, 2025.
- [49] Maohao Shen, Guangtao Zeng, Zhenting Qi, Zhang-Wei Hong, Zhenfang Chen, Wei Lu, Gregory Wornell, Subhro Das, David Cox, and Chuang Gan. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *arXiv preprint arXiv:2502.02508*, 2025.
- [50] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

- [51] Yexin Wu, Zhuosheng Zhang, and Hai Zhao. Mitigating misleading chain-of-thought reasoning with selective filtering. *arXiv preprint arXiv:2403.19167*, 2024.
- [52] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.
- [53] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.
- [54] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [55] George S Boolos, John P Burgess, and Richard C Jeffrey. *Computability and logic*. Cambridge university press, 2002.
- [56] B Jack Copeland. The church-turing thesis. 1997.
- [57] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.
- [58] Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*, 1, 2024.

A Task Introduction to Dataset

The dataset consists of purely synthetic data, covering 8 major algorithms and 28 tasks, as shown in Table A1. Most tasks have approximately 100,000 training samples and 500 test samples (a small subset of tasks, which are more difficult to construct, retain 20,000 training samples and 200 test samples). All test queries have been verified and are not included in the training set.

Algorithm	Task Name	Task Content
Simulation	Large Number Addition	$x_1 + x_2$ * ($\text{len}(x_1) = n, \text{len}(x_2) = m$)
	Large Number Division	$x_1 \div x_2$ * ($\text{len}(x_1) = n, \text{len}(x_2) = m$)
	Bubble Sort	Bubble sort list of n non-repeat numbers
	Word Flip	Flip a sentence containing n letters
	Compare Numbers	Compare x_1 and x_2 * ($\text{len}(x_1) = n, \text{len}(x_2) = m$)
Recursion	Any Substring	Find all substrings of given string with length n
	Derangement	Derangement count for n elements
Iteration	Population Growth	Calculate total pairs after n units, starting reproduction at x -th unit ($x < n$) with initial y pairs, z pairs produced per unit ($y, z \in \mathbb{N}^+$)
Greedy	Dijkstra	Shortest path values in graph with n vertices
	Multi-Machine Schedule	Maximum benefit of n tasks on x queues ($x < n$)
Enumeration	Count Letters	Count letters in sentence of length n
	Diophantine Equation	Find integer solutions to $x_1a + x_2b = n$ ($a, b \geq 0$)
	Prime Search	All prime numbers in the interval $[n, m]$ ($n < m$)
DP	0-1 Knapsack	Maximum benefit of n -element 0-1 knapsack
	LCS	LCS of string X_1 and X_2 ($\text{len}(X_1) = n, \text{len}(X_2) = m$)
	Levenshtein Distance	Minimum operations converting string X_1 to X_2
Divide & Conquer	Binary Search	Binary search index in list of n increasing numbers
Backtracking	Permutation Combination	Number of combinations in n -element list (step by step)

Table A1: Dataset synthesised under instruction of TAIL, containing 8 algorithms and 18 tasks. n and m represent length in a given range $G \in \{S, M, L\}$. * indicates that a decimal point can be inserted in any bit of the operand in spific task.

B Chain-of-Thought Format Examples

Take a binary search task in Divide & Conquer algorithm as an example and show chain-of-thought (CoT) data sample of TAIL. Figure B1 is the query as direct input to LLMs. Figure B2 is the minimalist reasoning chains of TAIL, which contain only core modules. Their ability to generalize to longer sequences has been verified in each algorithm. Figure B3 is a kind of more user-friendly CoT of TAIL, which verifies the length generalization property on all 28 tasks.

Query: (Example 1)

Determine the index of -5259 in the sorted list $[-5957, -5259, -4195, -2263, 1289, 3514, 3632, 4284, 5991, 6578, 7333]$ with binary search (start from 0).

Query: (Example 2)

Find the index of -5259 in the sorted list $[-5957, -5259, -4195, -2263, 1289, 3514, 3632, 4284, 5991, 6578, 7333]$, starting from 0 .

Query: (Example 3)

Provide the 0-based binary search index for -5259 in $[-5957, -5259, -4195, -2263, 1289, 3514, 3632, 4284, 5991, 6578, 7333]$.

Figure B1: Example query belonging to the Divide & Conquer algorithm (Binary Search).

Minimalist reasoning chain of TAIL:

```

<Atomic Subtask> {0,10}
<Data Review> [(s0 = -5957), (s1 = -5259), (s2 = -4195), (s3 = -2263), (s4 = 1289), (s5 = 3514),
(s6 = 3632), (s7 = 4284), (s8 = 5991), (s9 = 6578), (s10 = 7333)]
<Data Review> Find = -5259
Mid = 5, s5 = 3514 > Find
<Linear Expansion> {0,10} → {0,4}

<Atomic Subtask> {0,4}
<Data Review> [(s0 = -5957), (s1 = -5259), (s2 = -4195), (s3 = -2263), (s4 = 1289)]
<Data Review> Find = -5259.
Mid = 2, s2 = -4195 > Find
<Linear Expansion> {0,4} → {0,1}

<Atomic Subtask> {0,1}
<Data Review> [(s0 = -5957), (s1 = -5259)]
<Data Review> Find = -5259
Mid = 0, s0 = -5957 < Find
<Linear Expansion> {1,1} → {1,1}.

<Atomic Subtask> {1,1}
<Data Review> [(s1 = -5259)]
<Data Review> Find = -5259
Mid = 1, s1 = -5259 = Find
<Linear Expansion> {1,1} → F

Final Answer: Index 1.

```

Figure B2: Minimalist reasoning chain of TAIL, with only core modules aligned with Turing Machines.

User-friendly CoT of TAIL:

Let's perform the binary search step by step.

1. The currently processed interval is $[-5957, -5259, -4195, -2263, 1289, 3514, 3632, 4284, 5991, 6578, 7333]$.
First we map each number to its index:

- Original index 0: -5957
- Original index 1: -5259
- Original index 2: -4195
- Original index 3: -2263
- Original index 4: 1289
- Original index 5: 3514
- Original index 6: 3632
- Original index 7: 4284
- Original index 8: 5991
- Original index 9: 6578
- Original index 10: 7333

The left boundary index is 0 and the right boundary index is 10.

Median index $\lfloor (0 + 10)/2 \rfloor = 5$, the median number is 3514 .

The target number -5259 is less than the median number 3514 , continue these operations on the left subinterval $[-5957, -5259, -4195, -2263, 1289]$.

2. The currently processed interval is $[-5957, -5259, -4195, -2263, 1289]$.

First we map each number to its index:

- Original index 0: -5957
- Original index 1: -5259
- Original index 2: -4195
- Original index 3: -2263
- Original index 4: 1289

The left boundary index is 0 and the right boundary index is 4.

Median index $\lfloor 0 + 4 \rfloor = 2$, the median number is -4195 .

The target number -5259 is less than the median number -4195 , continue these operations on the left subinterval $[-5957, -5259]$.

3. The currently processed interval is $[-5957, -5259]$.

First we map each number to its index:

- Original index 0: -5957
- Original index 1: -5259

The left boundary index is 0 and the right boundary index is 1.

Median index $\lfloor 0 + 1 \rfloor = 0$, the median number is -5957 .

The target number -5259 is greater than the median number -5957 , continue these operations on the right subinterval $[-5259]$.

4. The currently processed interval is $[-5259]$.

First we map each number to its index:

- Original index 1: -5259

The left boundary index is 1 and the right boundary index is 1.

Median index $\lfloor 1 + 1 \rfloor = 1$, the median number is -5259 .

The target number -5259 is equal to the median number -5259 , the search ends.

The target number -5259 is located at index 1.

Figure B3: User-friendly CoT of TAIL, improving human readability on top of core modules

C Data Proportion Study

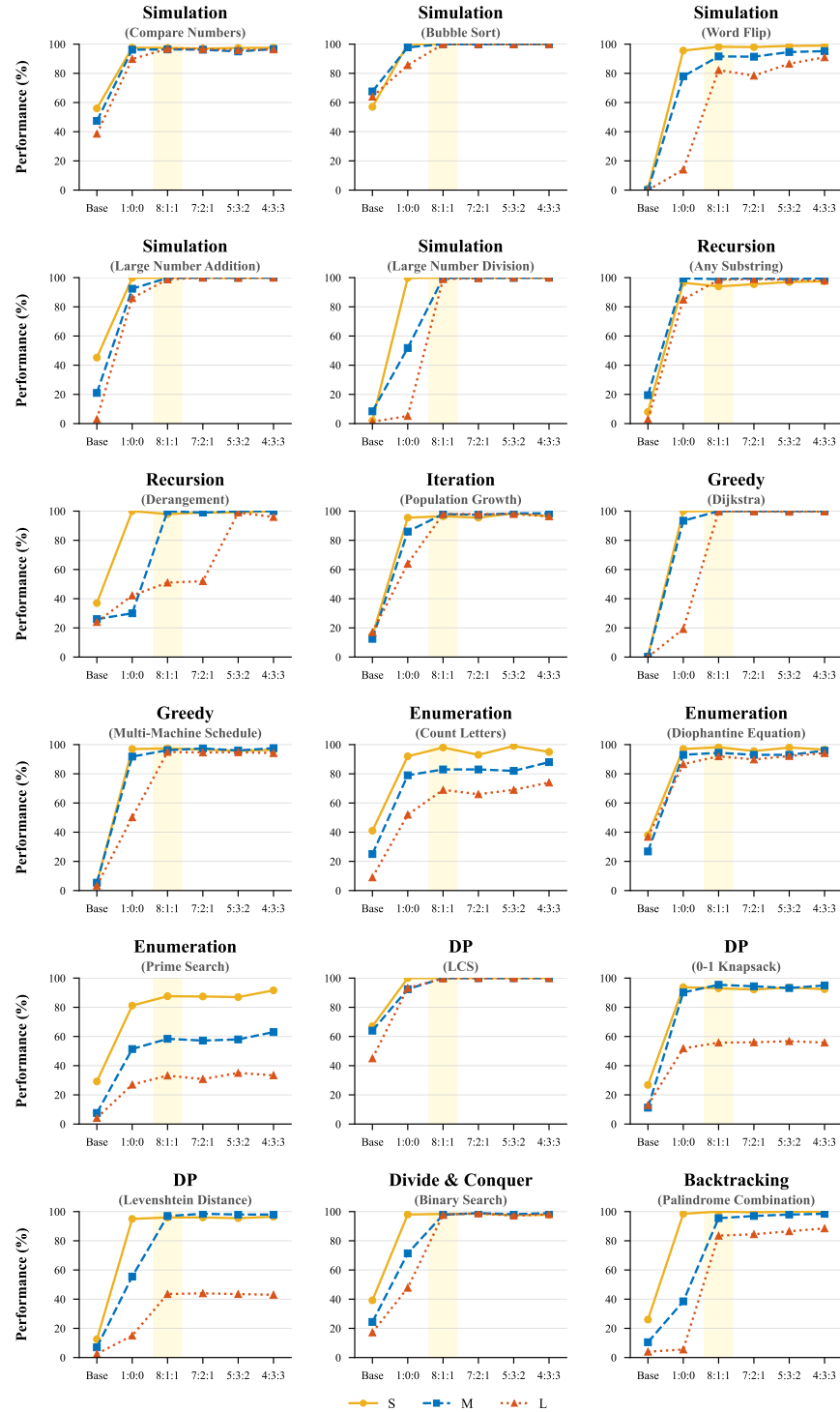


Figure C1: Mixed scale experiments with data on 18 tasks. We find that adding a small amount of long data to most unsaturated tasks achieves fast performance gains.

D Task Length Range

Algorithm	Task Name	Small (S)	Medium (M)	Long (L)
Simulation	Large Number Addition	[10, 30]	[31, 40]	[41, 50]
	Large Number Division	[2, 5]	[6, 10]	[11, 20]
	Bubble Sort	[2, 4]	[5, 6]	[7, 8]
	Word Flip	[10, 20]	[21, 50]	[51, 100]
	Compare Numbers	[5, 10]	[11, 20]	[21, 50]
Recursion	Any Substring	[3, 5]	[6, 9]	[10, 14]
	Derangement	[3, 30]	[31, 60]	[61, 100]
Iteration	Population Growth	[1, 10]	[11, 25]	[26, 50]
Greedy	Dijkstra	[3, 5]	[6, 10]	[11, 20]
	Multi-Machine Schedule	[5, 10]	[11, 20]	[21, 50]
Enumeration	Count Letters	[2, 6]	[7, 10]	[11, 20]
	Diophantine Equation	[10, 50]	[51, 100]	[101, 200]
	Prime Search	[5, 100]	[101, 200]	[201, 300]
DP	0-1 Knapsack	[2, 3]	[4, 5]	[6, 8]
	LCS	[2, 6]	[7, 9]	[10, 12]
	Levenshtein Distance	[2, 4]	[5, 7]	[8, 9]
Divide & Conquer	Binary Search	[5, 20]	[21, 40]	[41, 70]
Backtracking	Palindrome Combination	[2, 4]	[5, 6]	[7, 8]

Table D1: The length ranges across all tasks. The specific definition of ‘length’ for each task is detailed in Table A1, where length range $G \in \{S, M, L\}$.

E Baseline Data Construction

This section describes the data construction methods for the baseline methods (*i.e.*, Index Hint and Reversed Format). The experimental task was Large Number Addition (Simulation algorithm), and unlike the experiments in previous works, in this paper we contain a random number of decimals. Therefore, we improved the data construction method accordingly based on the principles of these baselines.

E.1 Index Hint

The Index Hint[8, 10] method refers to adding a hint of indexes to the corresponding numeric or logical bits of two operands for positioning in arithmetic or parity operations. This method has been extensively proven to be effective in both tasks. To compare the performance of Index Hint and TAIL, we refer to the method [8] that displays the indexes to locate as follows:

$$3a6b1c+5a7b6c=9a3b7c$$

However, the above approach is for the case where the two operands have the same number of digits and do not contain decimals, so we make the following improvement for non-fixed-length Index Hints that contain decimals:

$$1(-c)2(-b)3(-a).4(a)5(b)+6(-b)7(-a).8(a)9(b)=1(-c)9(-b)1(-a).3(a)4(b)$$

E.2 Reversed Format

Reversed Format [8, 10–13, 23] refers to reversing each of the two operands in arithmetic operations such as addition. The rationale for this method is that the addition is usually performed from the first digit, *i.e.*, from right to left. However, the order of next token prediction (NTP) in large language models (LLMs) is from left to right, which leads to overly complex search paths during model learning and affects the length generalization performance. This method is also a widely proven effective length generalization facilitation method, constructed as follows:

(Origin) $123.45+67.89=191.34$
 (Reversed Format) $54.321+98.76=43.191$

F Attention visualization of Memory Fetcher

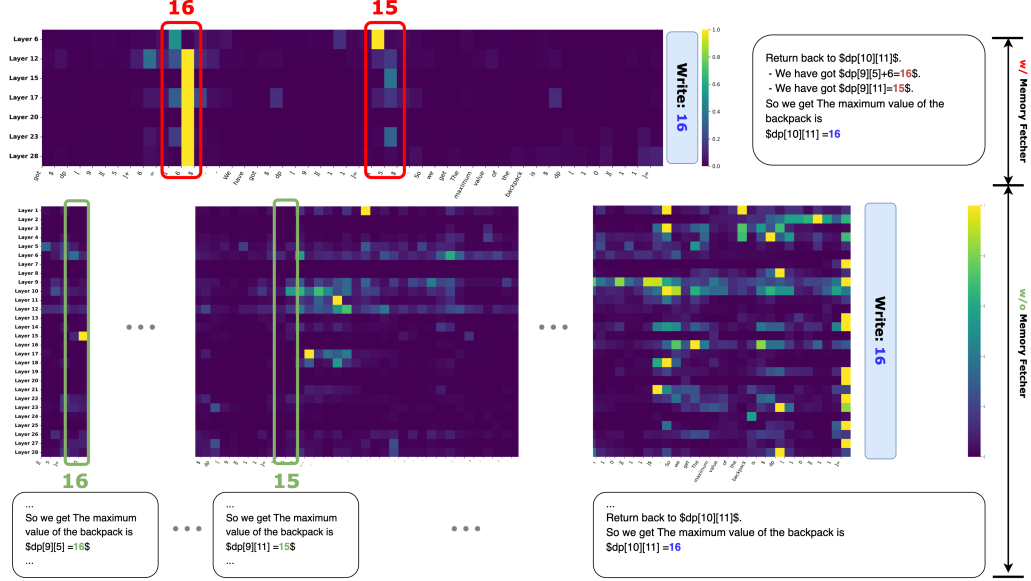


Figure F1: Ablation study on attention visualization of Memory Fetcher.

G Task Generalization Results

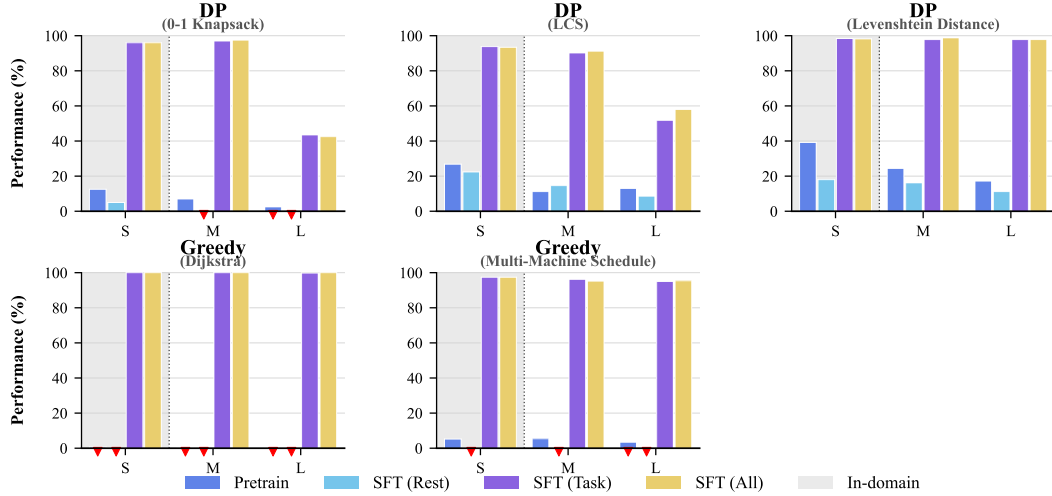


Figure G1: Generalization performance between tasks within a single algorithm (e.g., DP and Greedy). **Pretrain** represents Qwen 2.5-7B as the basis for subsequent SFT. For each task, **SFT(Rest)** indicates training using data from other tasks within the algorithm, **SFT(Task)** indicates training using data from this task, and **SFT(All)** indicates training using data from all tasks within this algorithm. ▼ indicates that this piece of data has a label accuracy of less than 5%.