LLM COMPLEXITY: TOWARDS A COMPUTATIONAL COMPLEXITY VIEW OF LARGE LANGUAGE MODEL INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Computational complexity theory offers a principled framework for characterizing the efficiency of computational problems, and Large Language Models (LLMs) can similarly be viewed as computational systems executing a structured process of input, inference-time compute, and output. In this context, the number of generated tokens emerges as a natural and directly observable proxy for inference-time compute. However, we show that the tempting analogy between LLM token length and classical time complexity does not hold. We disprove the hypothesis that inference-time token usage aligns monotonically with classical time complexity, demonstrating a fundamental divergence grounded in how LLMs actually allocate computation during inference. To this end, we introduce LLM Complexity, a new metric designed to quantify problem complexity from the perspective of LLMs by characterizing the area bounded above by the accuracy–token curve. We find that diverse LLMs induce highly consistent task rankings under LLM Complexity, indicating that the metric captures stable notions of difficulty across models. We leverage this property to construct an easy-to-hard curriculum for supervised fine-tuning, which improves accuracy while reducing token consumption. Finally, we demonstrate that LLM Complexity generalizes to real-world tasks, producing more stable cross-model correlations than human-annotated difficulty labels.

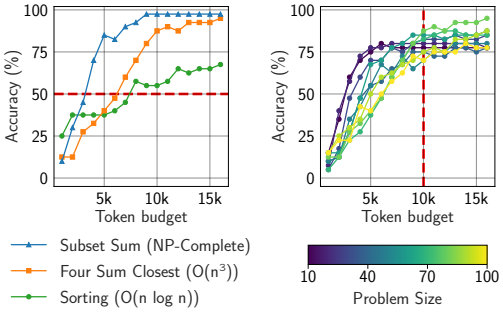
1 INTRODUCTION

Computational complexity theory lies at the core of computer science, offering a principled framework to characterize the efficiency and intrinsic properties of computational problems (Cook, 1971; Karp, 1972; Arora & Barak, 2009). Traditionally, complexity is defined in terms of the asymptotic growth of the required computational resources, *e.g.*, time complexity, as a function of problem size, measured by the number of elementary computational steps (Hartmanis & Stearns, 1965; Sipser, 1996). Large language models (LLMs), despite their architectural and operational differences from classical algorithms, can also be viewed as computational systems executing a structured process of the form: "input $\rightarrow$ inference-time compute $\rightarrow$ output." This perspective naturally raises a fundamental question: to what extent *can the behavior and capabilities of LLMs be analyzed through the lens of computational complexity theory?*

Addressing this question requires defining an appropriate notion of "elementary operations", analogous to those used in time complexity, for inference-time compute in LLMs. During generation, an LLM produces tokens autoregressively, with each token representing an intermediate computational state that later steps condition on (Vaswani et al., 2017; Brown et al., 2020). As a result, the number of generated tokens emerges as a natural and directly observable proxy for the inference-time compute. Moreover, a growing body of evidence suggests that allocating more inference-time compute—often manifested as longer reasoning traces—enables LLMs to solve a broader class of tasks (Jaech et al., 2024; Comanici et al., 2025; Zhang et al., 2026). These observations invite a tempting analogy: *does LLM token length play a role analogous to time complexity in classical computation?*

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this work, we show that this analogy may not hold. Specifically, we challenge the hypothesis that inference-time token usage monotonically aligns with classical worst-case time complexity through two counterexamples. First, we compare LLM accuracy and token usage across classical algorithmic problems. As shown in Figure 1a, *sorting*, an $O(n \log n)$ problem, requires more tokens to achieve 50% accuracy than the NP-hard *subset sum*. This likely arises because *sorting* requires long, structured outputs that amplify error propagation, whereas *subset sum* can often be solved through localized reasoning. Second, increasing problem size does not necessarily lead to monotonic performance degradation (Figure 1b). In search-based problems, LLMs can often exploit heuristic reasoning to focus on a small subset of plausible candidates, making performance only weakly dependent on input size. Together, these findings suggest that classical worst-case time complexity alone does not adequately characterize inference-time token usage in LLMs. This motivates an operational notion of complexity grounded in autoregressive inference, where generated tokens serve as the primitive computational step.



(a) Token usage misaligns with worst-case time complexity.

(b) Inference difficulty is not monotonic with problem size.

Figure 1: **Time complexity does not align with LLMs' inference-time token usage.** We conduct experiments on NEMOTRON-253B and report the relationship between the token budget and accuracy. (a) For problem size 80, *sorting* ($O(n \log n)$) requires more tokens to reach 50% accuracy than some tasks with higher theoretical complexity. (b) For *four sum closest*, accuracy at a 10K-token budget does not decrease monotonically with problem size. Together, these results highlight a mismatch between algorithmic complexity and LLM inference behavior.

To this end, we introduce LLM Complexity, a new metric designed to quantify problem complexity from the perspective of autoregressive large language models. Rather than relying on classical computational complexity, we characterize each problem by its accuracy–token curve, which captures performance as a function of a token budget. Intuitively, problems that achieve high accuracy under tight token constraints should be considered less complex, while those requiring substantially more tokens to reach comparable performance are more complex. Formally, we define LLM Complexity as the area bounded above by the accuracy–token curve: a smaller area corresponds to lower complexity, whereas a larger area indicates higher complexity due to greater requirements of inference-time compute. This formulation yields a problem complexity measure that is both model-aware and operationally meaningful, and reveals a striking property: despite differences in architecture and scale, various LLMs tend to induce highly consistent rankings under LLM Complexity.

We empirically evaluated this cross-model correlation by ranking twelve algorithmic tasks across five LLMs from three distinct model families. Using Spearman's ρ and Kendall's τ_b, we find that LLM Complexity rankings exhibit stronger agreement across models than rankings derived from common behavioral heuristics such as peak accuracy. Also, we find that LLM Complexity is robust to prompt variations, indicating that it captures stable task-level inference complexity rather than superficial signals *e.g.*, output token length. Exploiting this consistency, we demonstrate a practical application: complexity estimates obtained from a larger model can be used to construct an easy-to-hard curriculum for supervised fine-tuning a smaller model, yielding higher accuracy while reducing token consumption compared to random data ordering. We also extend LLM Complexity to real-world tasks on DeepMath He et al. (2026) and RuleArena (Zhou et al., 2025), and show that it yields more stable cross-model correlations than human-annotated difficulty labels, suggesting that LLM Complexity more faithfully captures how LLMs allocate inference-time compute during problem solving.

2 RELATED WORK

LLMs as computational systems. Strobl et al. (2024) models Transformers and LLMs as computational systems and analyzes their expressivity using tools from computation and formal language theory. Pérez et al. (2021); Bhattamishra et al. (2020); Jiang et al. (2026) further establishes computa-

tional universality. Unlike these studies, we focus on an operational notion of inference-time cost of LLMs and analyze it through the lens of computational complexity theory.

LLM reasoning and inference-time compute. LLM reasoning methods show that allocating more inference-time compute can substantially improve LLM performance. Chain-of-thought (Wei et al., 2022) prompting elicits explicit intermediate steps, and self-consistency scales inference-time compute by sampling multiple reasoning paths and aggregating to the most consistent answer (Wang et al., 2023). Recent work further explores search/refinement-based scaling of inference-time compute, *e.g.*, Mind Evolution (Lee et al., 2025b), alongside studies of redundant "overthinking" and budget-aware allocation (Chiang & Lee, 2024; Chen et al., 2025; Damani et al., 2025). In contrast, we quantify how accuracy scales under a bounded token budget and distill this relationship into a task-level complexity score for cross-task comparison and curriculum design.

On problem difficulty for LLMs. Prior work defines difficulty either *externally*, from observed success across a population of solvers (Ding et al., 2024; Zhou et al., 2026), or *internally*, as model-perceived hardness inferred from a target LLM's hidden representations (Zhu et al., 2025). This treats the LLM as a computational system and characterizes difficulty through a complexity-based lens, using inference-time token consumption as an operational resource measure to derive a cross-model-consistent notion of task difficulty.

3 TOKEN USAGE VS. TIME COMPLEXITY

When LLMs are viewed as computational systems, tokens provide a directly observable proxy for inference-time compute. This raises the hypothesis that token usage should increase monotonically with the computational time complexity of a problem. To test this, we evaluate five LLMs on twelve algorithmic tasks spanning diverse complexity classes and show that token usage does not correlate with worst-case time complexity across tasks.

3.1 EXPERIMENTAL SETUP

Data construction. To enable controlled and comparable scaling analysis, we consider a suite of classical algorithmic problems, ranging from $O(n \log n)$ *sorting* to NP-hard *subset sum*. Each problem takes a list of length n as input, where n is the problem size. For evaluation, we generate 32 instances per size using task-specific generators with standardized formats, varying $n \in 10, 20, \ldots, 100$. The full task list and prompts are provided in Section A.

Table 1: **Weak alignment between worst-case algorithmic time complexity and LLM token usage.** We conduct the experiment with 12 algorithmic problems, each evaluated across 10 problem sizes. For each model, we compute the correlation between the average output length and a task-level time-complexity ranking derived from tight, widely accepted worst-case bounds (*e.g.*, *sorting* with $O(n \log n)$). Across all five models, both Spearman's ρ and Kendall's τ_b remain small, indicating weak alignment between inference-time token usage and worst-case algorithmic time complexity.

Model	Spearman's ρ / Kendall's τ_b
NEMOTRON-49B	0.102 / 0.103
NEMOTRON-253B	0.008 / 0.098
GPT-OSS-20B	0.213 / 0.165
GPT-OSS-120B	0.213 / 0.165
QWEN-235B	0.188 / 0.132

Models. We evaluate five open-weight LLMs: LLAMA-3.3-NEMOTRON-SUPER-49B-V1 and LLAMA-3.1-NEMOTRON-ULTRA-253B-V1 from the Nemotron family Bercovich et al. (2025), GPT-OSS-20B and GPT-OSS-120B from the gpt-oss family Agarwal et al. (2025), and QWEN-235B from the Qwen3 family Yang et al. (2025). We focus on open-weight models as our evaluation requires access to the full generated reasoning trace, where most closed-source APIs, *e.g.*, GPT, Claude, Gemini, expose only partial or summarized reasoning traces.

Note that all the experiments are conducted in a text-only setting. Models are not permitted to use external tools, programs, or execution environments; this is enforced via prompt design and, when available, model-specific configurations.

3.2 CORRELATION BETWEEN TOKEN USAGE & TIME COMPLEXITY

To test whether token usage reflects worst-case time complexity, we prompt each LLM to solve each task instance step by step and record the generated tokens. For each problem, we average output length across sizes, rank tasks by this statistic, and compare it to rankings induced by worst-case time complexity using Spearman's ρ and Kendall's τ_b.

The Results in Table 1 show uniformly low correlations across all five models, indicating that token usage is weakly aligned with worst-case time complexity. To further analyze this, we report average output length per task in Section C.1. We make two observations. First, *four sum*, *four sum in range*, and *three sum in range* show low token usage, as these search-style tasks enable heuristic reasoning without exhaustive exploration. Second, *sorting*, despite its $O(n \log n)$ complexity, incurs high token usage due to long structured outputs that increase generation length and error propagation.

Overall, the results suggest that token usage reflects LLM-specific inference behaviors rather than algorithmic worst-case time complexity, highlighting the need for a complexity notion grounded in how LLMs allocate computation during inference.

4 Approach

Since time complexity does not directly characterize LLM reasoning behavior, we seek an effective metric to quantify task complexity under generation constraints with the following desiderata:

1. **Cross-model agreement.** The metric should induce consistent complexity rankings across various models and across algorithmic and real-world tasks, rather than model-dependent behaviors.
2. **Prompt robustness.** The metric should remain stable under different prompt variations, rather than being driven by prompt-dependent behaviors, *e.g.*, verbosity or reasoning style.
3. **Learning utility.** The metric should provide practical value for downstream applications, such as constructing more effective and efficient curriculum learning strategies.

To this end, we describe our protocol for measuring performance across varying token budgets (Section 4.1), and then formalize LLM Complexity, our proposed metric, as a scalar measure of the computational cost required to achieve high accuracy (Section 4.2).

4.1 Calculating accuracy across token budgets

To consistently measure performance across budgets, we adopt a prefix-truncation procedure Lee et al. (2025a); Ghosal et al. (2025); Muennighoff et al. (2025), ensuring that differences reflect marginal compute rather than sampling variance. We define token budgets $\mathcal{B} = 1{,}000, \ldots, 16{,}000$ and generate a single long reasoning trace per instance with $b_{\max} = 16{,}000$, using continuation prompts if needed (Section B.2). Following "budget forcing" (Muennighoff et al., 2025), we truncate the trace at each $b \in \mathcal{B}$ and prompt the model to answer from the partial trace.

We evaluate different truncations of the same reasoning process across budgets, using task accuracy at each budget b. Per-instance correctness is defined as $\mathbf{1}[\hat{y} = y]$, where y is the ground truth and $\hat{y} = \hat{y}(x; c, n, b)$ is the final answer under budget b. For each problem c and size n, instances $x \sim \mathcal{D}_{c,n}$, and accuracy at budget b is the expected correctness: $\mathrm{Acc}(c, n, b) = \mathbb{E}_{x \sim \mathcal{D}_{c,n}} \left[\mathbf{1}\left[\hat{y}(x; c, n, b) = y(x; c, n)\right] \right]$. Evaluating $\mathrm{Acc}(c, n, b)$ over the budget set $\{b_1, \ldots, b_K\}$ yields the performance measurements used in the following section.

4.2 Defining LLM Complexity

A robust characterization of task complexity should reflect how accuracy changes as the available token budget varies. Therefore, we evaluate each task under multiple token budgets (Section 4.1) and define LLM Complexity by aggregating the resulting budget–accuracy measurements.

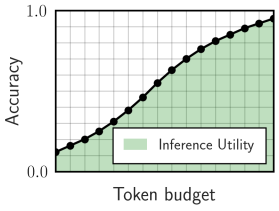


Figure 2: **Illustration of inference utility.** We plot accuracy as a function of token budget. The shaded area under the curve represents aggregate accuracy across the budget.

As illustrated in Figure 2, we summarize each task's budget–accuracy scaling behavior using the **inference utility**, defined as the area under the accuracy–budget curve over $\mathcal{B}$. We compute this area using the trapezoidal rule in log-budget space: $\mathcal{U}(c, n) = \sum_{i=1}^{N-1} \frac{\mathrm{Acc}(c,n,\log b_i) + \mathrm{Acc}(c,n,\log b_{i+1})}{2} \cdot (\log b_{i+1} - \log b_i)$. Using log-budget space prevents a large token budget from disproportionately dominating the aggregate.

To facilitate comparison across tasks and problem sizes, we normalize by the log-budget interval length: $\bar{\mathcal{U}}(c, n) = \frac{\mathcal{U}(c,n)}{\log b_{\max} - \log b_{\min}}, \quad \bar{\mathcal{U}}(c, n) \in [0, 1]$, where $b_{\min} = b_1$ and $b_{\max} = b_N$.

Table 2: **Cross-model correlation of task-level complexity rankings.** We report Spearman's ρ / Kendall's τ_b between the per-task complexity ranks induced by Model A and Model B (higher is better) on 12 algorithmic problems. **Length**: The number of tokens in the model's first phase CoT trace. **Budget**: the token budget required to reach 90% of the model's peak accuracy, normalized by the peak accuracy. **Peak**: the model's peak accuracy. The results show that **LLM Complexity** yields the best cross-model correlation among all the other heuristic complexity definitions.

Model A	Model B	Length	Budget	Peak	LLM Complexity
NEMOTRON-49B	NEMOTRON-253B	**0.926** / 0.777	0.881 / 0.746	0.860 / 0.743	0.919 / 0.768
NEMOTRON-49B	GPT-OSS-20B	0.711 / 0.518	**0.778** / **0.609**	0.661 / 0.568	0.761 / 0.549
NEMOTRON-49B	GPT-OSS-120B	**0.850** / **0.657**	0.807 / 0.648	0.716 / 0.655	0.800 / 0.614
NEMOTRON-49B	QWEN-235B	0.720 / 0.507	0.650 / 0.511	0.792 / **0.686**	**0.809** / 0.630
NEMOTRON-253B	GPT-OSS-20B	0.738 / 0.538	0.782 / 0.614	0.693 / 0.622	**0.837** / **0.649**
NEMOTRON-253B	GPT-OSS-120B	**0.864** / 0.668	0.818 / 0.673	0.766 / 0.626	0.840 / **0.687**
NEMOTRON-253B	QWEN-235B	**0.838** / 0.621	0.711 / 0.578	0.804 / **0.723**	0.810 / 0.634
GPT-OSS-20B	GPT-OSS-120B	0.872 / 0.694	0.867 / 0.768	0.858 / 0.744	**0.913** / **0.782**
GPT-OSS-20B	QWEN-235B	**0.894** / **0.721**	0.708 / 0.582	0.684 / 0.610	0.780 / 0.606
GPT-OSS-120B	QWEN-235B	**0.948** / **0.800**	0.722 / 0.613	0.691 / 0.645	0.874 / 0.720
Average		**0.836** / 0.650	0.772 / 0.634	0.752 / 0.662	0.834 / **0.664**

Finally, we define the LLM Complexity of task c at problem size n as $\mathcal{C}(c,n) = 1 - \bar{\mathcal{U}}(c,n)$, so larger values of $\mathcal{C}(c,n)$ indicate that achieving high accuracy requires a greater inference-time token budget. LLM Complexity provides a cross-task comparable notion of complexity under bounded generation. It is based on inference utility, which aggregates accuracy across multiple token budgets in log-budget space. We examine its key properties in the following sections. Moreover, LLM Complexity is insensitive to tokenizer choice: small variations typically induce near-uniform rescaling of token counts, which does not materially affect the resulting rankings.

5 EXPERIMENTAL RESULTS

To determine whether LLM Complexity captures an effective complexity signal, we first map the *topology of complexity* across tasks and problem sizes (Section 5.1). Then, we evaluate its **cross-model agreement** through correlations across different models and tasks (Section 5.2). Finally, we assess its **prompt robustness**, examining whether the metric remains stable under prompt variations rather than being driven by superficial prompt-dependent behaviors such as verbosity or reasoning style (Section 5.3). Note that we focus on open-weight models as our evaluation requires access to the full generated reasoning trace, where most closed-source APIs, *e.g.*, GPT, Claude, Gemini, expose only partial or summarized reasoning traces.

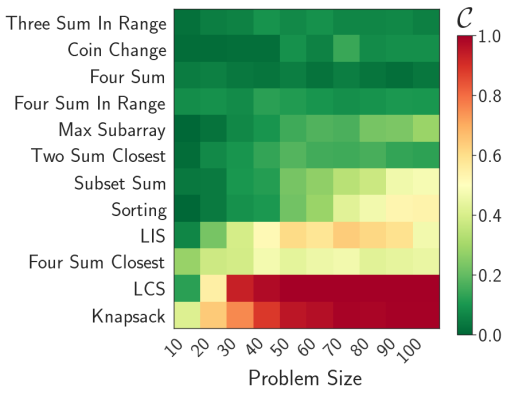


Figure 3: **LLM Complexity heatmap across algorithmic tasks and problem sizes.** The heatmap shows the averaged LLM Complexity ($\mathcal{C}$) of different algorithmic problems and corresponding problem size over 5 models (NEMOTRON-49B, NEMOTRON-253B, GPT-OSS-20B, GPT-OSS-120B, QWEN-235B). The results show that tasks with lower average complexity have stable scores across problem sizes, whereas tasks with higher average complexity tend to show increasing complexity as problem size increases.

5.1 LLM COMPLEXITY ACROSS TASKS

To characterize the topology of task complexity, Figure 3 shows the average LLM Complexity across tasks and problem sizes. Tasks with lower complexity exhibit stable scores across input lengths, whereas higher-complexity tasks show increasing complexity as problem size grows. This suggests that harder tasks require tracking and updating more intermediate states, leading to higher inference-time compute.

Table 3: **Prompt robustness of task-level complexity rankings.** We report the average Spearman's ρ / Kendall's τ_b between the rankings induced by each prompt variant and all other prompt settings across five models (higher is better). LLM Complexity achieves the highest correlations in 5 out of 6 settings and also attains the highest average correlation, indicating strong robustness to prompt variations. In particular, the **Length** baseline drops substantially under the no_step setting, suggesting greater sensitivity to verbosity-related prompting effects.

Prompt Variants	Length	Budget	Peak	LLM Complexity
default_setting	0.873 / 0.748	0.895 / 0.802	0.863 / 0.802	**0.931 / 0.849**
no_step	0.762 / 0.634	0.866 / 0.787	0.867 / 0.807	**0.928 / 0.854**
be_concise	0.896 / 0.791	0.900 / 0.821	0.848 / 0.796	**0.933 / 0.871**
short_continue	0.910 / 0.808	0.903 / 0.828	0.863 / 0.809	**0.937 / 0.880**
long_continue	0.916 / 0.808	0.883 / 0.802	0.883 / 0.833	**0.937 / 0.876**
answer_format	**0.892 / 0.774**	0.774 / 0.692	0.751 / 0.703	0.811 / 0.754
Average	0.875 / 0.761	0.870 / 0.789	0.846 / 0.792	**0.913 / 0.847**

To quantify agreement across models, we next study cross-model consistency by correlating the task rankings induced by LLM Complexity across LLMs.

5.2 Cross-model agreement

A robust complexity metric should capture task-level inference difficulty that is consistent across different models, rather than behaviors specific to a particular model. If **LLM Complexity** is valid, different LLMs should agree on the relative complexity ranking of tasks, even if their absolute performance differs. We benchmark our metric against three baselines to isolate this signal: **Length:** The raw number of reasoning tokens generated (proxy for consumption). **Peak:** The maximum performance achievable regardless of budget (proxy for capability). **Budget:** The normalized token budget required to reach 90% of peak performance (proxy for efficiency). We quantify cross-model agreement using rank correlations: Spearman's ρ and Kendall's τ_b, computed over the global ordering of all problem–size pair instances.

Results in Table 2 yield two insights. First, LLM Complexity captures the most stable complexity signal, achieving the highest average Kendall's τ_b and competitive Spearman's ρ, suggesting it filters out model-specific artifacts present in length-based metrics. Second, we observe a **family clustering effect**: within-family pairs (e.g., Nemotron) show higher correlations than cross-family ones, indicating that while neural hardness is largely universal, pre-training induces subtle "dialects" of complexity that LLM Complexity can capture.

5.3 Prompt robustness

A good metric should not only remain stable across models but also capture stable task-level differences in inference difficulty rather than superficial verbosity. If the metric were driven primarily by verbosity, its rankings would fluctuate under different prompting strategies. Therefore, it should remain stable under prompt variations. To examine this, we evaluate five prompt variants (full prompts are provided in Section B.3):

- **no_step**: Remove the "step-by-step" reasoning instruction (Kojima et al., 2022).
- **be_concise**: Instruct the LLM to be concise during reasoning (Renze & Guven, 2024).
- **short_continue**: Replace the default continuation prompt with a shorter version that does not explicitly encourage self-verification.
- **long_continue**: Replace the default continuation prompt with a longer version that encourages deeper reflection and rethinking.
- **answer_format**: Modify the instruction to require list outputs with one item per line (Long et al., 2025).

First, we evaluate whether the rankings induced by different prompt variants remain consistent with one another. For each prompt setting, we report its average Spearman's ρ and Kendall's τ_b with all other settings over 12 tasks at input length 100, averaged across five models, as shown in Table 3; the full pairwise results are provided in Section C.2.1. The correlations remain consistently high across

Table 4: **Accuracy usage under different curricula.** We fine-tune QWEN3-1.7B with teacher-generated supervision from NEMOTRON-253B using 3 different seeds. All curriculum variants use the same sampling form $P \propto \exp(-\mathcal{C}/\tau)$ with an annealed temperature to induce an easy-to-hard schedule, differing only in the choice of complexity score $\mathcal{C}$: **Time** (wall-clock execution time), **Length**, **Budget** (defined in Table 2), and **LLM Complexity**; **Random** uses uniform sampling, and **Pre-trained** denotes the base model without SFT. **LLM Complexity** achieves the highest average accuracy and lowest average token usage, supporting its effectiveness for curriculum design.

Method	Seen	Unseen
Pre-traine	$42.75 \pm 0.00\%$	$46.25 \pm 0.00\%$
Time	$38.92 \pm 0.80\%$	$41.25 \pm 2.17\%$
Length	$44.17 \pm 1.01\%$	$45.42 \pm 4.02\%$
Budget	$43.33 \pm 0.52\%$	$43.75 \pm 3.31\%$
Random	$41.75 \pm 2.63\%$	$41.67 \pm 1.44\%$
LLM Complexity	$\mathbf{44.42 \pm 0.76\%}$	$\mathbf{48.33 \pm 5.05\%}$

variants, indicating that LLM Complexity is robust to prompt variations and captures stable, model-consistent differences in task complexity rather than prompt-dependent artifacts. In contrast, other baselines—particularly the Length baseline—exhibit noticeably weaker and less reliable correlations, reflecting their greater sensitivity to prompt-induced verbosity effects. The table in Section C.2.2 further shows that LLM Complexity achieves the strongest cross-model agreement across prompt settings compared with the baselines in Table 3.

In summary, our empirical results establish LLM Complexity as a reliable "thermometer" for neural reasoning complexity. It captures a signal that is both grounded in task structure and consistently recognized across models and prompt settings. Having validated the fidelity of this signal, we next investigate its **learning utility** for optimizing the learning process.

6 CURRICULUM LEARNING

As **cross-model agreement** and **prompt robustness** of LLM Complexity are shown in Section 5, a natural question arises: can this signal provide **learning utility**? Prior work suggests that easy-to-hard curricula can improve learning efficiency Bengio et al. (2009); Lee et al. (2024), but their effectiveness heavily depends on the quality of the difficulty metric. We therefore construct a curriculum based on LLM Complexity and evaluate its effect on fine-tuning performance.

6.1 COMPLEXITY-BASED CURRICULUM SAMPLING

Given a training instance from problem c at size n with normalized complexity $\mathcal{C}(c,n) \in [0,1]$, we assign it a Boltzmann-style score $s(c,n) = \exp(-\mathcal{C}(c,n)/\tau)$, where $\tau > 0$ controls the sampling preference over difficulty levels. Smaller τ concentrates sampling on low-complexity (easy) instances, while larger τ produces a flatter distribution and increases coverage of higher-complexity (hard) instances. To avoid vanishing probabilities, we smooth the score as $\tilde{s}(c,n) = (1-\epsilon)s(c,n) + \frac{\epsilon}{N}$, where $\epsilon \geq 0$ is a smoothing constant and N is the number of problem–size pairs. We then normalize over all problem–size pairs: $P(c,n) = \frac{\tilde{s}(c,n)}{\sum_{(c',n')} \tilde{s}(c',n')}$. Smaller τ concentrates sampling on low-complexity (easy) instances, while larger τ produces a flatter distribution and increases coverage of higher-complexity (hard) instances.

6.2 CURRICULUM SCHEDULER

To form an easy-to-hard curriculum, we bias training toward low-complexity instances early on and gradually increase the probability of sampling more challenging instances as training progresses. Under the previous sampling scheme, this behavior can be achieved by annealing the temperature τ during the T training steps, where we linearly increase τ as $\tau(t) = \tau_0 + \frac{t-1}{T-1}(\tau_1 - \tau_0), \quad t \in \{1, \ldots, T\}$, where $\tau_0 = \tau(1)$ denotes the initial temperature at the start of training and $\tau_1 = \tau(T)$ denotes the final temperature at the end of training.

6.3 CURRICULUM LEARNING RESULTS

We evaluate whether LLM Complexity can provide a reliable curriculum signal for SFT. Concretely, we fine-tune QWEN3-1.7B on a mixture of problem–size instances (c, n) sampled from the complexity-based distribution in the previous paragraph.

Training data and supervision. For each training instance, we use a reasoning trace generated by NEMOTRON-253B as the supervision signal (*i.e.*, a teacher-produced chain-of-thought / "thinking" trace paired with the final answer). We then perform SFT on QWEN3-1.7B to learn to reproduce the teacher-style reasoning and answers.

Curriculum vs. baselines. We compare data-ordering strategies under identical SFT settings (Section E). All curricula share the same weighted sampling form $P_{\tau(t)}(\cdot)$ (Section 6.1) and temperature annealing schedule $\tau(t)$, differing only in the per-instance difficulty score used in the weights:

- **Pre-trained** uses QWEN3-1.7B pre-trained weight.
- **Random SFT** uniformly samples instances from the training pool.
- **LLM Complexity curriculum** uses $\mathcal{C}(c, n)$, *i.e.*, our LLM Complexity score.
- **Length curriculum** uses the number of tokens in the supervision signal as difficulty.
- **Time curriculum** measures the task execution time in a Linux environment as difficulty.
- **Budget curriculum**: difficulty is the compute-budget proxy defined in Section 5.2.

Results and analysis.

Table 4 summarizes accuracy under different curricula using 3 different seeds. The SFT training set contains ten *seen* algorithm families: *sorting, max subarray, coin change, two sum closest, subset sum, four sum, four sum in range, LCS, knapsack*, and *four sum closest*. We hold out two *unseen* algorithms (*LIS* and *three sum in range*) to evaluate generalization beyond the training distribution. We also verify that SFT exhibits the expected seen–unseen gap and meaningful adaptation; see Section E.5 for details. Overall, the LLM Complexity curriculum outperforms all other curricula on the *seen* algorithms, indicating that complexity provides an effective signal for organizing SFT. It also improves accuracy on the *unseen* algorithms, suggesting that the learned reasoning patterns transfer beyond the training set.

7 FITTING LLM COMPLEXITY TO COMPUTATIONAL COMPLEXITY

We investigate how LLM Complexity relates to traditional complexity metrics by predicting its ranking from three measures: *output length*, *space complexity*, and *time complexity*. Output length captures the expected answer length; for example, sorting outputs all n elements, whereas four sum outputs only four. For each model, we fit non-negative weights (α, β, γ) summing to one over the three induced rankings. We compute $s_c = \alpha r_c^{\text{out}} + \beta r_c^{\text{space}} + \gamma r_c^{\text{time}}$ and select the weights that maximize Spearman's ρ between the ranking induced by s_c and the LLM Complexity ranking.

Table 5 reports the resulting mixture weights. Overall, output length consistently receives substantial weight, suggesting that it is informative for algorithmic task complexity. While weights vary across models, models within the same family exhibit similar patterns, possibly reflecting shared reasoning strategies, such as similar reliance on intermediate states. Beyond this linear decomposition, we further analyze pairwise correlations in Section D.

8 GENERALIZING TO REAL-WORLD PROBLEMS

To examine if our LLM Complexity goes beyond synthetic algorithmic problems and generalizes to real-world tasks, we conduct experiments on DeepMath He et al. (2026) and RuleArena (Zhou et al., 2025), which were released after the knowledge cutoff of all evaluated models (*i.e.*, no data contamination issue).

DeepMath. DeepMath He et al. (2026) contains math problems with difficulty labels assigned by GPT-4o (Hurst et al., 2024) using Art of Problem Solving guidelines (AoPSOnline, 2009) to approximate human-perceived difficulty. We evaluate 200 problems, comparing LLM Complexity rankings across models and against these difficulty labels. As shown in Table 6, LLM Complexity rankings exhibit substantial cross-model agreement, consistent with Section 5, but near-zero correlation with

Table 5: **Fitted problem-level mixture weights over computational complexity measures.** Let r_c^{out}, r_c^{space}, and r_c^{time} denote rankings induced by output length, auxiliary-space complexity, and time complexity, respectively, and r_c^{model} the LLM Complexity ranking. Output length refers to the task-implied output length rather than the model's actual response length. We grid-search nonnegative weights (α, β, γ) summing to one and compute $s_c = \alpha r_c^{\text{out}} + \beta r_c^{\text{space}} + \gamma r_c^{\text{time}}$, selecting the weights that maximize Spearman's ρ between the induced ranking and r_c^{model}. This provides a simple decomposition of LLM Complexity across the three proxies.

Model	α (Output)	β (Space)	γ (Time)
NEMOTRON-49B	0.49	0.51	0.00
NEMOTRON-253B	0.48	0.51	0.01
GPT-OSS-20B	0.55	0.25	0.20
GPT-OSS-120B	0.58	0.23	0.19
QWEN-235B	0.56	0.25	0.19

Table 6: **Task-level complexity correlation on DeepMath.** We report cross-model correlation (averaged per model over pairwise comparisons) and correlation with DeepMath difficulty annotations using Spearman's ρ and Kendall's τ_b. LLM Complexity shows substantial cross-model consistency but near-zero correlation with human-annotated difficulty, suggesting that it captures a complementary dimension of LLM-specific problem complexity. Full pairwise correlations are provided in Section C.3.

Model	Cross-model	vs. DeepMath
NEMOTRON-49B	0.544 / 0.482	-0.063 / -0.051
NEMOTRON-253B	0.500 / 0.451	-0.071 / -0.059
GPT-OSS-20B	0.460 / 0.423	0.001 / 0.001
GPT-OSS-120B	0.478 / 0.441	-0.002 / -0.002
QWEN-235B	0.546 / 0.491	-0.111 / -0.093

Table 7: **Cross-model consistency of task-level complexity rankings on RuleArena.** We report Spearman's ρ / Kendall's τ_b between the rankings of the nine RuleArena tasks induced by Model A and Model B (higher is better). LLM Complexity achieves the strongest cross-model agreement.

Model A	Model B	Length	Budget	Peak	LLM Complexity
NEMOTRON-49B	NEMOTRON-253B	0.983 / 0.944	0.983 / 0.939	0.897 / 0.818	**1.000 / 1.000**
NEMOTRON-49B	GPT-OSS-20B	0.817 / 0.722	0.868 / 0.765	**0.945 / 0.883**	0.881 / 0.783
NEMOTRON-49B	GPT-OSS-120B	0.633 / 0.556	**0.966 / 0.899**	0.573 / 0.418	**0.966 / 0.899**
NEMOTRON-49B	QWEN-235B	0.800 / 0.611	0.828 / 0.758	0.879 / 0.758	**1.000 / 1.000**
NEMOTRON-253B	GPT-OSS-20B	0.833 / 0.778	**0.928 / 0.824**	0.843 / 0.706	0.881 / 0.783
NEMOTRON-253B	GPT-OSS-120B	0.617 / 0.500	0.932 / 0.841	0.735 / 0.597	**0.966 / 0.899**
NEMOTRON-253B	QWEN-235B	0.750 / 0.556	0.897 / 0.818	0.983 / 0.939	**1.000 / 1.000**
GPT-OSS-20B	GPT-OSS-120B	0.683 / 0.500	0.862 / **0.761**	0.498 / 0.377	**0.867** / 0.722
GPT-OSS-20B	QWEN-235B	0.383 / 0.333	**0.894** / 0.765	0.826 / 0.647	0.881 / **0.783**
GPT-OSS-120B	QWEN-235B	0.467 / 0.389	0.729 / 0.667	0.684 / 0.537	**0.966 / 0.899**
Average		0.697 / 0.589	0.889 / 0.804	0.786 / 0.668	**0.941 / 0.877**

DeepMath's labels. This suggests that human-oriented mathematical difficulty and LLM inference-time complexity capture different aspects of problem difficulty.

RuleArena. RuleArena (Zhou et al., 2025) contains real-world reasoning tasks across three categories—airline baggage fees, NBA transactions, and tax regulations—with three difficulty levels each. We treat each category–difficulty pair as a task and evaluate cross-model agreement over the resulting nine tasks. As shown in Table 7, LLM Complexity achieves the highest average rank correlation across model pairs and ranks within the top two in most comparisons, demonstrating strong cross-model consistency on real-world reasoning tasks.

9 CONCLUSION

We propose LLM Complexity, a metric that quantifies problem complexity for LLMs via the area over the accuracy–token curve. We show that LLM Complexity captures stable task-level differences in inference difficulty, yielding rankings that are highly consistent across diverse model families. We further demonstrate its practical utility by constructing easy-to-hard curricula for supervised fine-tuning that improve accuracy while reducing token consumption. Finally, we show that LLM Complexity generalizes to real-world tasks, producing more stable cross-model correlations than human-annotated difficulty labels. Overall, our findings highlight the importance of defining complexity in terms of how LLMs allocate inference-time compute, rather than relying solely on classical algorithmic hierarchies.

10 AI USE STATEMENT

We used generative AI tools to assist with writing and polishing the manuscript, primarily to improve clarity, readability, and language. We also used generative AI tools for retrieval and discovery, such as identifying potentially relevant literature. All AI-assisted content and identified references were reviewed and verified by the authors. We take full responsibility for the final content of the paper.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

AoPSOnline. Aops wiki: Competition ratings. `https://artofproblemsolving.com/wiki/index.php/AoPS_Wiki:Competition_ratings`, 2009. Accessed: 2026-01-28.

Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

Yoshua Bengio, JérÃ´me Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, et al. Llama-nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*, 2025.

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. On the computational power of transformers and its implications in sequence modeling. In *Conference on Computational Natural Language Learning*, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In *International Conference on Machine Learning*, 2025.

Cheng-Han Chiang and Hung-Yi Lee. Over-reasoning and redundant calculation of large language models. In *European Chapter of the Association for Computational Linguistics*, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 1971.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *International Conference on Learning Representations*, 2025.

Mucong Ding, Chenghao Deng, Jocelyn Choo, Zichu Wu, Aakriti Agrawal, Avi Schwarzschild, Tianyi Zhou, Tom Goldstein, John Langford, Animashree Anandkumar, et al. Easy2hard-bench: Standardized difficulty labels for profiling llm performance and generalization. *Advances in Neural Information Processing Systems*, 37, 2024.

Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking more always help? mirage of test-time scaling in reasoning models. In *Advances in Neural Information Processing Systems*, 2025.

Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117, 1965.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. In *International Conference on Learning Representations*, 2026.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Hongjian Jiang, Michael Hahn, Georg Zetzsche, and Anthony Widjaja Lin. Softmax transformers are turing-complete. In *International Conference on Learning Representations*, 2026.

Richard M. Karp. *Reducibility among Combinatorial Problems*. Springer US, 1972.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, 2022.

Ayeong Lee, Ethan Che, and Tianyi Peng. How well do LLMs compress their own chain-of-thought? a token complexity approach. *arXiv preprint arXiv:2503.01141*, 2025a.

Bruce W Lee, Hyunsoo Cho, and Kang Min Yoo. Instruction tuning with human curriculum. In *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper LLM thinking. *arXiv preprint arXiv:2501.09891*, 2025b.

Do Xuan Long, Ngoc-Hai Nguyen, Tiviatis Sim, Hieu Dao, Shafiq Joty, Kenji Kawaguchi, Nancy F. Chen, and Min-Yen Kan. LLMs are biased towards output formats! systematically evaluating and mitigating output format bias of LLMs. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Empirical Methods in Natural Language Processing*, 2025.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22, 2021.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, 2024.

Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

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

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Duzhen Zhang, Zhong-Zhi Li, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Xiuyi Chen, Yingying Zhang, Fei Yin, Jiahua Dong, Zhijiang Guo, Le Song, and Cheng-Lin Liu. From system 1 to system 2: A survey of reasoning large language models. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 48, 2026.

Hongli Zhou, Hui Huang, Ziqing Zhao, Lvyuan Han, Huicheng Wang, Kehai Chen, Muyun Yang, Wei Bao, Jian Dong, Bing Xu, et al. Lost in benchmarks? rethinking large language model benchmarking with item response theory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Ruiwen Zhou, Wenyue Hua, Liangming Pan, Sitao Cheng, Xiaobao Wu, En Yu, and William Yang Wang. Rulearena: A benchmark for rule-guided reasoning with llms in real-world scenarios. In *Association for Computational Linguistics*, 2025.

Yubo Zhu, Dongrui Liu, Zecheng Lin, Wei Tong, Sheng Zhong, and Jing Shao. The LLM already knows: Estimating LLM-perceived question difficulty via hidden representations. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

A PROBLEM TABLE

Table 8 provides a complete specification of the evaluation suite. For each problem, we list the problem name and include a representative instance prompt drawn from the same template used during evaluation, illustrating the exact input format (*e.g.*, list-structured inputs or paired strings) and the required output type (*e.g.*, an ordered list, a subset, a subsequence, or a scalar objective value). All problems are classical algorithmic problems with verifiable outputs, enabling consistent automatic checking across instances. As described in our data construction procedure, we instantiate each problem at multiple problem sizes (from 10 to 100 in increments of 10), so Table 8 serves as a concrete reference for how each problem is presented to the model while the evaluation varies the underlying problem scale. Table 9 reports the annotated worst-case time and auxiliary-space complexity of the 12 tasks. We use these annotations to form the task-level time and space complexity rankings used in our experiments.

B PROMPT DETAILS

B.1 PROMPT USED FOR INFERENCE

Our inference pipeline consists of two stages: (1) long-chain generation and (2) answer generation from a truncated prefix.

Long-chain generation. To ask the LLM generate the long chain, we use the following prompt:

```
{Problem Specific Prompt} Provide your reasoning step by step
without using external programming functions.  Lastly, present
the final answer in the exact format:  Answer:  [...]
```

Answer generation from a truncated prefix. To ask the LLM to generate an answer based on the truncated prefix, we use the following prompt:

```
{Problem Specific Prompt} Compute the result manually without
using external programming functions.  Present the final
answer in the exact format:  Answer:  [...]
```

The `{Problem Specific Prompt}` varies across problems. For example, for *sorting*, one instance is:

```
Given the list [36, 23, 34, 73, 98], sort the list in
ascending order.
```

We provide a sample prompt for each task in Appendix A.

B.2 CONTINUATION PROMPT

If the LLM terminates generation before reaching the maximum token budget, we append the following prompt to encourage it to continue reasoning until the budget is exhausted.

```
Please review your previous reasoning step by step and
check for any errors, incorrect assumptions, or logical
inconsistencies.  If issues are found, revise the reasoning
accordingly.
```

B.3 DETAILED PROMPTS FOR PROMPT SENSITIVITY EXPERIMENTS

In Section 5.3, we show that LLM Complexity is robust to prompt variations. Here, we detail the prompt variants used in our experiments.

- **no_step**: For long-chain generation, we replace the default prompt with:

```
Compute the result manually without using external
programming functions.  Do not provide any reasoning
or explanation.  Lastly, present the final answer in the
exact format:  Answer:  ...
```

- **be_concise**: For long-chain generation, we use the following prompt:

```
Provide your reasoning step by step without using
external programming functions.  Lastly, present the
final answer in the exact format:  Answer:  [...]  Be
concise.
```

- **short_continue**: We replace the default continuation prompt with:

```
Please continue thinking.
```

- **long_continue**: We replace the default continuation prompt with:

```
You have previously generated a detailed reasoning
process for this problem.  Now, carefully review your
entire reasoning step by step.  For each step, check
whether the logic is valid, whether any assumptions are
incorrect, and whether any intermediate calculations or
conclusions are flawed.  If you identify any mistakes,
inconsistencies, or gaps in reasoning, revise the
reasoning accordingly and ensure that the corrected
version is logically coherent and consistent with the
problem statement.  After completing this verification
and correction process, provide the final answer.
Ensure that the final answer is correct and consistent
with your revised reasoning.
```

- **answer_format**: For answer generation, we replace the default prompt with:

```
Compute the result manually without using external
programming functions.  generate your list such that
each element is in a new line, and present the final
answer in the exact format:\n\nAnswer:\na\nb\nc\nd
```

Table 8: Tasks used in our evaluation.

Task	Sample Prompt
Max Subarray	Given the list nums = [-38, 41, 24, -42, 46], find the maximum possible sum of a contiguous subarray.
Sorting	Given the list [36, 23, 34, 73, 98], sort the list in ascending order.
Two Sum Closest	Given a list of integers of length 5: [25, 182, 149, 17, 193], find two numbers whose sum is closest to the target value 31. If multiple answers exist, return any one of them.
Longest Increasing Subsequence (LIS)	Given the list [5, 4, 3, 1, 2], find the longest increasing subsequence.
Three Sum (In Range)	Given a list of integers of length 5: [118, 46, 52, 110, 107], find any 3 numbers whose sum is in the range [268, 276] (inclusive).
Four Sum	Given the list of integers of length 5: [96, 58, 60, 56, 50], find any four numbers that sum to 264.
Four Sum (In Range)	Given the list of integers of length 5: [107, 20, 60, 99, 146], find any 4 numbers whose sum is in the range [282, 287] (inclusive).
Four Sum Closest	Given the list of integers of length 5: [131, 19, 173, 12, 187], find four numbers whose sum is closest to the target value 449. There is no exact solution where the sum equals 449. If multiple answers achieve the same closest distance, return any one of them.
Coin Change	Given coin denominations coins = [1, 5, 9, 11, 14, 18, 25, 26, 27] (unlimited supply of each), find the minimum number of coins needed to make exactly amount = 11. Return the minimum number of coins as an integer.
Subset Sum	Given the list [96, 11, 82, 59, 54] and the target sum 302, find a subset of the list that sums exactly to the target.
Longest Common Subsequence (LCS)	Given two strings s1 and s2, find a longest common subsequence. s1 = ejbpc s2 = kefiq Return any one LCS string (if multiple exist, any is acceptable).
Knapsack (0/1)	You are given n items. items = [(weight:18,value:3), (weight:11,value:7), (weight:4,value:17), (weight:18,value:8), (weight:15,value:17)] Capacity W = 26 Choose a subset of items (each item at most once) to maximize total value without exceeding capacity.

Table 9: **Annotated time and auxiliary-space complexity for the 12 algorithmic tasks.** Time bounds follow the tightest widely accepted *worst-case* complexity under the standard computation model. Space reports worst-case *auxiliary* memory, excluding input/output storage. Here n is input list length; for Longest Common Subsequence, n, m are the two sequence lengths; for Coin Change, A is the target amount and k is the number of coin types; for Knapsack, W is capacity; for Subset Sum, T is the target sum. †Assuming in-place sorting; recursion stack is $O(\log n)$ if applicable. ‡Subset Sum is NP-complete in n; we annotate the exponential worst-case in n while noting its pseudo-polynomial DP dependence on T.

Task	**Time**	**Aux. Space**	**Notes / Parameters**
Max Subarray	$O(n)$	$O(1)$	Kadane scan
Sorting	$O(n \log n)$	$O(1)^{\dagger}$	Comparison-based sorting
Two Sum Closest	$O(n \log n)$	$O(1)^{\dagger}$	Sort + two pointers
LIS	$O(n \log n)$	$O(n)$	Patience sorting
Three Sum (In Range)	$O(n^2)$	$O(1)^{\dagger}$	Sort + two pointers
Four Sum	$O(n^3)$	$O(1)^{\dagger}$	Sort + 2 loops + two pointers
Four Sum (In Range)	$O(n^3)$	$O(1)^{\dagger}$	Same as Four Sum
Four Sum Closest	$O(n^3)$	$O(1)^{\dagger}$	Same as Four Sum
Coin Change	$O(kA)$	$O(A)$	DP over amount A; k coin types
LCS	$O(nm)$	$O(\min(n, m))$	DP time; Hirschberg space
Knapsack (0/1)	$O(nW)$	$O(W)$	DP over capacity W
Subset Sum	$O(2^n)^{\ddagger}$	$O(T)^{\ddagger}$	NP-complete; pseudo-polynomial DP in target T

Table 10: Average number of output tokens for each problem under step-by-step prompting, aggregated across all problem sizes and LLMs, sorted from low to high.

Task	Avg. output tokens
Four Sum	1968
Four Sum In Range	2851
Three Sum In Range	2854
Coin Change	3404
Subset Sum	3508
Two Sum Closest	4094
Max Subarray	4411
Sorting	5466
LIS	7907
LCS	10505
Four Sum Closest	10926
Knapsack	12421

C Additional Results

C.1 Task-level average output length

For each task, we measure the length of the model's generated output in tokens and report the task-level average. Concretely, for every problem, we aggregate token counts across all evaluated instances and problem sizes under the same decoding configuration. We report the results in Table 10.

C.2 Additional prompt sensitivity results

Here, we provide additional results from the prompt sensitivity experiments in Section 5.3, further demonstrating that LLM Complexity is robust to prompt variations. Specifically, Section C.2.1 presents the full pairwise correlations between different prompt settings, providing a more fine-grained view of prompt-level agreement. We then analyze cross-model consistency under each prompt variant in Section C.2.2, showing that LLM Complexity maintains strong agreement across diverse models even under substantial prompt changes.

C.2.1 Pairwise prompt sensitivity results

To complement the aggregate prompt-sensitivity results in Table 3, we provide the full pairwise correlations between all prompt settings in Table 11.

For each prompt setting, we report its Spearman's ρ and Kendall's τ_b with every other setting over 12 tasks at input length 100, averaged across five models.

The Average row within each block corresponds to the aggregate value reported in Table 3, while the Overall Average row summarizes agreement across all prompt pairs.

C.2.2 Cross-model consistency under prompt settings

We further evaluate cross-model consistency under each prompt variant by averaging Spearman's ρ and Kendall's τ_b across all model pairs. As shown in Table 12, LLM Complexity achieves the highest overall agreement among all metrics, with an average correlation of 0.903 / 0.780. It ranks first in three out of five settings and second in the remaining two, while the **Length** baseline drops notably under **no_step**, indicating higher sensitivity to prompt-induced verbosity changes.

C.3 Additional DeepMath results

We report the full pairwise cross-model correlations of task-level complexity rankings on DeepMath in Table 13. For each pair of models, we compute Spearman's ρ and Kendall's τ_b over instance-level

Table 11: **Detailed pairwise prompt sensitivity results.** For each prompt setting, we report its pairwise Spearman's ρ / Kendall's τ_b with every other setting over 12 tasks at input length 100, averaged across five models. The Average row within each block summarizes the mean agreement of that prompt setting with all others, while Overall Average averages across all prompt-setting pairs. Bold and underline denote the highest and second-highest values within each row, with Spearman's ρ and Kendall's τ_b ranked independently.

Prompt Setting	Compared Setting	Length	Budget	Peak	LLM Complexity
default_setting	no_step	0.751 / 0.612	0.887 / 0.783	0.893 / 0.828	**0.940 / 0.848**
	be_concise	0.885 / 0.763	0.904 / 0.805	0.853 / 0.779	**0.943 / 0.855**
	short_continue	0.919 / 0.806	0.934 / 0.867	0.871 / 0.800	**0.956 / 0.884**
	long_continue	0.929 / 0.812	0.899 / 0.803	0.943 / **0.898**	**0.962** / 0.885
	answer_format	**0.881** / 0.745	0.850 / 0.754	0.754 / 0.704	0.854 / **0.775**
	Average	0.873 / 0.748	0.895 / 0.802	0.863 / 0.802	**0.931 / 0.849**
no_step	default_setting	0.751 / 0.612	0.887 / 0.783	0.893 / 0.828	**0.940 / 0.848**
	be_concise	0.747 / 0.630	0.907 / 0.838	0.899 / 0.834	**0.970 / 0.895**
	short_continue	0.775 / 0.648	0.921 / 0.847	0.895 / 0.840	**0.967 / 0.901**
	long_continue	0.787 / 0.667	0.903 / 0.832	0.917 / 0.857	**0.966 / 0.895**
	answer_format	0.749 / 0.612	0.712 / 0.634	0.731 / 0.677	**0.796 / 0.728**
	Average	0.762 / 0.634	0.866 / 0.787	0.867 / 0.807	**0.928 / 0.854**
be_concise	default_setting	0.885 / 0.763	0.904 / 0.805	0.853 / 0.779	**0.943 / 0.855**
	no_step	0.747 / 0.630	0.907 / 0.838	0.899 / 0.834	**0.970 / 0.895**
	short_continue	0.945 / 0.860	0.958 / 0.886	0.863 / 0.823	**0.983 / 0.936**
	long_continue	0.966 / 0.879	0.944 / 0.863	0.898 / 0.861	**0.972 / 0.918**
	answer_format	**0.937 / 0.824**	0.786 / 0.712	0.727 / 0.683	0.798 / 0.750
	Average	0.896 / 0.791	0.900 / 0.821	0.848 / 0.796	**0.933 / 0.871**
short_continue	default_setting	0.919 / 0.806	0.934 / 0.867	0.871 / 0.800	**0.956 / 0.884**
	no_step	0.775 / 0.648	0.921 / 0.847	0.895 / 0.840	**0.967 / 0.901**
	be_concise	0.945 / 0.860	0.958 / 0.886	0.863 / 0.823	**0.983 / 0.936**
	long_continue	0.958 / 0.860	0.926 / 0.845	0.898 / 0.839	**0.977 / 0.921**
	answer_format	**0.954 / 0.867**	0.775 / 0.694	0.788 / 0.741	0.799 / 0.759
	Average	0.910 / 0.808	0.903 / 0.828	0.863 / 0.809	**0.937 / 0.880**
long_continue	default_setting	0.929 / 0.812	0.899 / 0.803	0.943 / **0.898**	**0.962** / 0.885
	no_step	0.787 / 0.667	0.903 / 0.832	0.917 / 0.857	**0.966 / 0.895**
	be_concise	0.966 / 0.879	0.944 / 0.863	0.898 / 0.861	**0.972 / 0.918**
	short_continue	0.958 / 0.860	0.926 / 0.845	0.898 / 0.839	**0.977 / 0.921**
	answer_format	**0.938 / 0.824**	0.745 / 0.665	0.758 / 0.710	0.807 / 0.758
	Average	0.916 / 0.808	0.883 / 0.802	0.883 / 0.833	**0.937 / 0.876**
answer_format	default_setting	**0.881** / 0.745	0.850 / 0.754	0.754 / 0.704	0.854 / **0.775**
	no_step	0.749 / 0.612	0.712 / 0.634	0.731 / 0.677	**0.796 / 0.728**
	be_concise	**0.937 / 0.824**	0.786 / 0.712	0.727 / 0.683	0.798 / 0.750
	short_continue	**0.954 / 0.867**	0.775 / 0.694	0.788 / 0.741	0.799 / 0.759
	long_continue	**0.938 / 0.824**	0.745 / 0.665	0.758 / 0.710	0.807 / 0.758
	Average	**0.892 / 0.774**	0.774 / 0.692	0.751 / 0.703	0.811 / 0.754
Overall Average		0.875 / 0.761	0.870 / 0.789	0.846 / 0.792	**0.913 / 0.847**

complexity scores induced by LLM Complexity. These results complement the aggregated statistics in the main text by providing a more fine-grained view of agreement across model families.

Overall, we observe moderate to strong positive correlations across most model pairs, indicating that LLM Complexity induces relatively consistent complexity rankings even across models with different architectures and scales.

Table 12: The average Spearman's ρ / Kendall's τ_b of complexity rankings across all model pairs under each prompt variant (higher is better). **LLM Complexity** achieves the strongest overall cross-model agreement, with the highest average correlation across prompt variations and consistently high performance across settings.

Ablation	Length	Budget	Peak	LLM Complexity
no_step	0.618 / 0.491	0.814 / 0.681	0.732 / 0.651	**0.905 / 0.785**
be_concise	0.888 / 0.758	0.887 / 0.763	0.725 / 0.643	**0.899 / 0.776**
short_continue	**0.923 / 0.806**	0.868 / 0.738	0.803 / 0.738	0.891 / 0.765
long_continue	**0.917 / 0.800**	0.852 / 0.728	0.752 / 0.663	0.912 / 0.794
answer_format	0.878 / 0.732	0.896 / **0.784**	0.716 / 0.647	**0.906** / 0.780
Average	0.845 / 0.717	0.863 / 0.739	0.746 / 0.668	**0.903 / 0.780**

Table 13: **Full pairwise cross-model correlation on DeepMath.** We report Spearman's ρ and Kendall's τ_b between task-level complexity rankings induced by LLM Complexity for all model pairs. Each entry is computed over instance-level complexity scores and reflects the agreement in relative task difficulty between two models. Higher values indicate stronger alignment. Consistent with the main results, correlations are generally moderate to strong, with higher agreement observed within model families and slightly lower (but still positive) correlations across different families.

Model A	Model B	Spearman ρ	Kendall τ_b
NEMOTRON-49B	NEMOTRON-253B	0.679	0.604
NEMOTRON-49B	GPT-OSS-20B	0.419	0.375
NEMOTRON-49B	GPT-OSS-120B	0.421	0.376
NEMOTRON-49B	QWEN-235B	0.655	0.574
NEMOTRON-253B	GPT-OSS-20B	0.357	0.324
NEMOTRON-253B	GPT-OSS-120B	0.346	0.315
NEMOTRON-253B	QWEN-235B	0.617	0.560
GPT-OSS-20B	GPT-OSS-120B	0.648	0.619
GPT-OSS-20B	QWEN-235B	0.414	0.376
GPT-OSS-120B	QWEN-235B	0.498	0.454

Notably, models within the same family (e.g., NEMOTRON or GPT-OSS) tend to exhibit higher agreement, while cross-family correlations are slightly lower but remain positive. This pattern suggests that while some variation exists due to model-specific behaviors, the proposed metric captures a shared notion of inference-time difficulty.

Table 14: **Grounding LLM Complexity in computation complexity proxies.** Each cell reports the correlation between the LLM Complexity ranking and a ranking induced by a classical measure (output length, space, or time), shown as Spearman's ρ / Kendall's τ_b.

Model	Output	Space	Time
Nemotron-49B	0.598 / 0.388	0.271 / 0.201	0.141 / 0.067
Nemotron-253B	0.537 / 0.356	0.166 / 0.101	0.137 / 0.101
Gpt-oss-20B	0.665 / 0.482	0.518 / 0.422	0.471 / 0.382
Gpt-oss-120B	0.606 / 0.440	0.356 / 0.253	0.362 / 0.287
Average	0.602 / 0.417	0.328 / 0.244	0.278 / 0.209

D THE CORRELATION OF LLM COMPLEXITY AND COMPUTATIONAL COMPLEXITIES

To better understand the characteristics of LLM Complexity, we compare LLM Complexity against two classical measures—space complexity and time complexity—and an additional proxy, *output length*, defined as the expected token length of the ground-truth answer.

For each problem, we construct problem-level rankings induced by these three measures and measure their rank consistency with each model's problem-level LLM Complexity ranking using Spearman's ρ and Kendall's τ_b. For each model, we first aggregate complexity scores across input lengths to obtain a single problem-level LLM Complexity score per problem, and apply the same aggregation procedure to each classic measure to obtain problem-level rankings. We measure space complexity and time complexity using worst-case auxiliary memory usage and worst-case runtime, respectively.

Table 14 reports the results. Overall, output length shows the strongest alignment with LLM Complexity ranking on average. One plausible explanation is that, in our benchmark, longer gold answers typically require producing more symbols and maintaining more intermediate information. For example, in sorting, the gold answer is an ordered list of n elements, so the output length grows with n. Generating this list autoregressively is more than "printing" n tokens: at each step, the model must maintain the partially sorted prefix and choose the next element to remain consistent with both the remaining inputs and the prefix, leading to more token usage. We also observe that the gpt-oss models correlate more strongly with space and time complexity than the Nemotron models, suggesting that gpt-oss complexity is more tightly coupled to problems that demand a larger maintained state or longer sequential computation.

E Supervised fine-tuning and inference details

E.1 Supervised fine-tuning (SFT)

Teacher supervision. We fine-tune Qwen3-1.7B on NVIDIA RTX A6000 with 48 GB VRAM using supervised traces produced by Nemotron-253B. For each training instance, we use the teacher's first-phase reasoning trace as the supervision target.

Chat formatting. Each example is serialized into a single text sequence using the model's chat template:

$$[\text{system}] \parallel [\text{user: prompt}] \parallel [\text{assistant: completion}],$$

with a fixed system prompt: "`Please solve the following algorithmic problem without using programming languages.`"

Parameter-efficient tuning (LoRA). We fine-tune using LoRA adapters with $r = 16$, $\alpha = 32$, dropout $= 0.05$, `bias=none`, and `target_modules=all-linear`. The base model is loaded with `torch_dtype=bfloat16` and `device_map=auto`, and we enable TF32 matmul for faster training.

Optimization hyperparameters. We train for 2 epochs with learning rate 2×10^{-4}, per-device batch size 1, and gradient accumulation 16 (effective batch size 16). We use a cosine learning-rate schedule with warmup ratio 0.03. The maximum sequence length is set to 4096 tokens.

E.2 Curriculum sampling used in SFT

Temperature-annealed weighted sampling. In curriculum mode, we sample training examples with replacement from a temperature-controlled distribution over examples:

$$P_t(i) \;\propto\; (1-\epsilon)\exp(-\mathcal{C}_i/\tau(t)) + \epsilon \cdot \frac{1}{N},$$

where $\mathcal{C}_i$ is the example complexity (from `difficulty.json`), N is dataset size, and ϵ is an exploration coefficient. We linearly anneal the temperature from $\tau_0 = 0.15$ to $\tau_1 = 0.6$ across training steps, with $\epsilon = 0.05$.

E.3 Inference and evaluation (vLLM)

Decoding configuration. We use greedy decoding by default (temperature $= 0.0$, top-$p = 1.0$) with repetition penalty 1.1. We set `max_new_tokens`=8192 and `max_model_len`=8192 to bound generation length and total context length.

E.4 KEY HYPERPARAMETERS (SUMMARY)

Table 15: SFT and inference configurations used in our experiments.

Component	Setting
SFT base model	QWEN3-1.7B
SFT supervision	NEMOTRON-253B trace
SFT epochs / LR	2 / 2×10^{-4}
SFT batch / accum	1 / 16
SFT max seq len	4096
SFT LR schedule	cosine, warmup ratio 0.03
LoRA	$r = 16$, $\alpha = 32$, dropout 0.05, `all-linear`
Curriculum	$\tau_0 = 0.15$, $\tau_1 = 0.6$, $\epsilon = 0.05$
Inference engine	vLLM (batched decoding)
Prompt mode	chat template, `enable_thinking=True`
Decoding	greedy ($T = 0$), top-$p = 1.0$, rep. penalty 1.1
Max tokens / context	`max_new_tokens`=8192, `max_model_len`=8192
vLLM runtime	bf16, `gpu_mem_util`=0.90, batch size 32
Eval filter	len $\in [10, 100]$, idx $\in [16, 19]$, len mod 10 = 0

E.5 EFFECTIVENESS OF SFT

Figure 4 validates that our SFT setup learns non-trivial signal from the teacher traces. Across almost all *seen* algorithms (left of the dashed line), the SFT model achieves substantially higher accuracy on *seen instances* (training split) than on *unseen instances* (held-out split), indicating that fine-tuning meaningfully adapts the base model to this problem family rather than producing random or degenerate outputs. Moreover, curriculum-based sampling consistently improves performance over random-order SFT on the held-out split for most algorithms, suggesting better generalization instead of merely overfitting to the training instances. Finally, on *unseen algorithms* (right of the dashed line), curriculum SFT yields noticeably stronger transfer than random-order SFT, supporting our claim that the curriculum improves robustness to distribution shift.

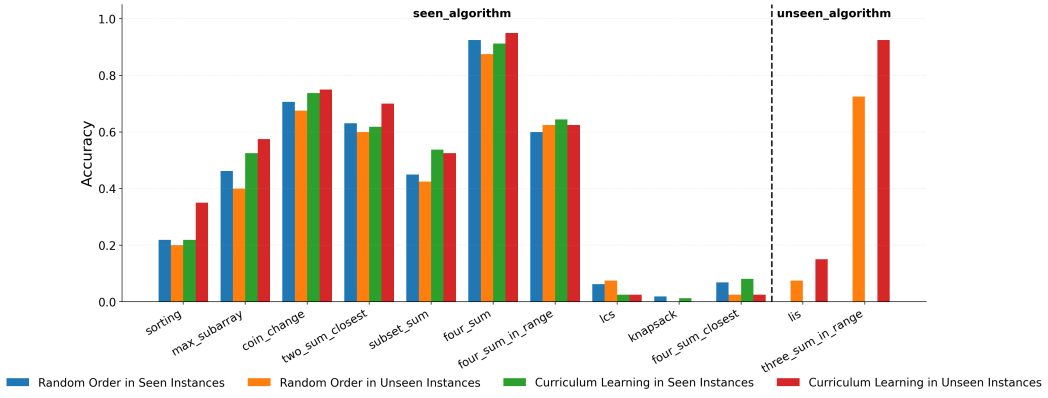


Figure 4: **SFT learns and curriculum improves generalization.** Accuracy on *seen* (training-split) vs. *unseen* (held-out) instances for each algorithm. Bars compare random-order SFT and curriculum SFT (complexity-based weighted sampling) evaluated on both splits. The dashed line separates algorithms included in SFT training (seen algorithms) from algorithms excluded from training (unseen algorithms). SFT yields higher accuracy on seen instances than unseen instances for seen algorithms, confirming effective adaptation, while curriculum SFT improves held-out performance and transfers better to unseen algorithms.