ALGORITHMIC DEPTH PREDICTS THE SAMPLE COMPLEXITY OF GENERALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

How much training data does a model need before it generalizes? We hypothesize that the answer depends on how difficult the task is, and test this hypothesis on compositional tasks, where task difficulty is the complexity of the algorithm that generates the data. Across thousands of trained models, sample complexity grows exponentially with algorithmic depth, the number of sequentially dependent steps, following a trend that predicts deeper tasks before training. However, depth describes the algorithm, not the learning problem. Supervising intermediate steps during training, while the model must still generate them at test time, reduces sample complexity by orders of magnitude for the same algorithm. We find that sample complexity is set by the unobserved algorithmic depth, the longest run of steps without supervision, so where supervision is placed matters more than how much is provided. The same holds for pretrained language models on English versions of our tasks, real programs, and GSM8K math word problems. Our results show that training data has a computational structure as well as a size, and point to a less discussed advantage of chain-of-thought training, which lowers sample complexity by shortening the reasoning a model must learn on its own.

1 INTRODUCTION

Much of the recent progress in deep learning has come from scaling, where larger models trained on more data improve in predictable ways [1, 2]. This progress depends on training data, and high-quality data is becoming limited [3, 4]. This makes us ask how much training data a model actually needs before it generalizes to unseen data, which is the question we study in this work. *We hypothesize that the answer depends on how difficult the task is, and that harder tasks need more data.* Prior work links the onset of generalization to properties of the model and of training, such as the point at which a model's memorization capacity is filled [5, 6], the length of training [7–9], or the number of distinct training tasks [10, 11]. Since none of them depends on how difficult the task is, they leave open why an easy task and a hard task would need different amounts of data [12, 13].

To test this hypothesis, we need tasks whose difficulty we can control and measure. We use compositional tasks, in which a small transformer applies a sequence of randomly generated operations to an input (Figure 1a). Because the data is generated by a known algorithm, task difficulty here is algorithmic complexity, the amount of computation needed to turn the input into the answer, which is also the computation the model has to learn. We measure algorithmic complexity with algorithmic depth (D), the number of sequentially dependent operations [14–16]. Because we know exactly which compositions appear in training, we can cleanly test generalization on unseen compositions, and we define sample complexity as the number of distinct training compositions a model needs before it generalizes. Therefore, *our central hypothesis is that algorithmic complexity predicts sample complexity.* Across more than 2,000 trained models and different kinds of operations, we find that sample complexity grows exponentially with algorithmic depth, and can therefore be predicted from algorithmic complexity before training. A single scaling trend fitted on shallow tasks predicts the sample complexity of deeper tasks, and of tasks made harder in other ways, without refitting.

However, algorithmic depth describes the algorithm, not the learning problem, and a learning problem can be made easier without changing the algorithm [17, 18]. One way to do this is to supervise intermediate steps, which is a controlled version of Chain-of-Thought (CoT) training and process supervision [19–21]. During training, the targets include the outputs of selected intermediate steps.

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.

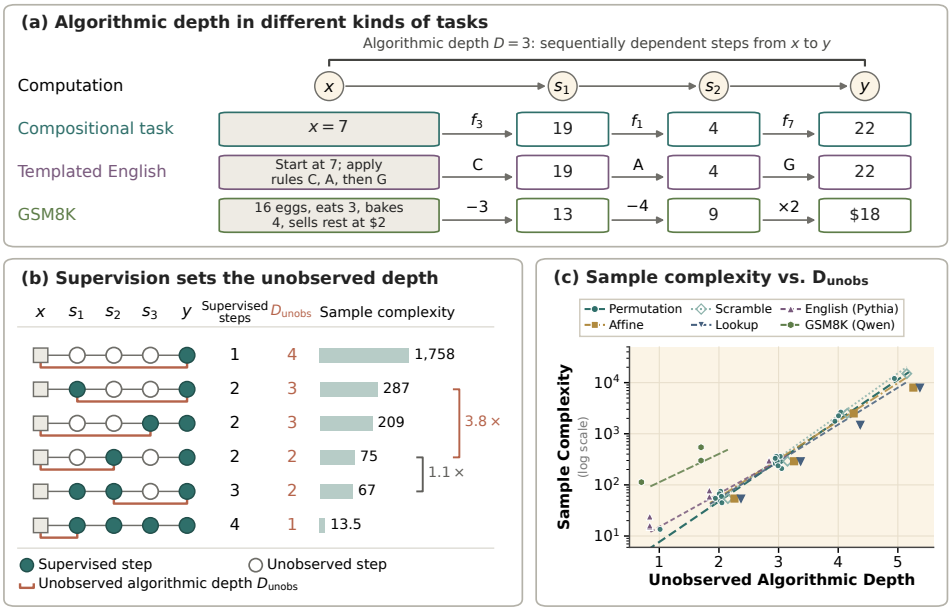

Figure 1: **Sample complexity depends on how much of the algorithm is learned without supervision.** **(a)** Algorithmic depth (D) is the number of sequentially dependent steps from input to answer. **(b)** Supervising intermediate steps sets the unobserved algorithmic depth (D_{unobs}), the longest run of steps between supervised outputs. **(c)** Across compositional and natural language tasks, sample complexity grows exponentially with D_{unobs}.

At test time, the model sees only the input and must generate these intermediate outputs itself before the final answer, as in standard CoT training. For the same algorithm, this reduces sample complexity by orders of magnitude. We find that sample complexity is set by the unobserved algorithmic depth (D_{unobs}), which we define as the longest run of steps the model must compute between two supervised outputs (Figure 1b). The same scaling trend, fitted without intermediate supervision and evaluated at D_{unobs} instead of D, predicts the sample complexity of every supervision scheme we test without refitting (Figure 1c). As a result, where supervision is placed matters more than how much is provided. Two learning problems for the same algorithm that supervise the same number of intermediate steps, but at different positions, can differ several-fold in sample complexity.

To test whether this trend holds beyond compositional tasks, we fine-tune a pretrained Pythia-70M [22] on the same tasks written in English, and Qwen2.5-3B [23] on GSM8K math word problems [24], supervising different steps of the human-written solutions. On GSM8K, hiding one reasoning step raises sample complexity from about 110 to 400 (3.5×), and hiding every step never reaches the same accuracy within 1,600. On the English tasks, each unobserved step costs 4–5× more data, while supervising every step keeps sample complexity nearly constant, at 15 to 24 compositions from three to five steps. These results points to a less discussed advantage of CoT training, which reduces the growth of sample complexity with depth from exponential to nearly constant.

Our main contributions are as follows:

- We study how sample complexity depends on task difficulty, showing that it grows exponentially with algorithmic depth across different kinds of operations, while model size decides only whether a task can be learned (§ 4).
- We introduce unobserved algorithmic depth, the longest run of steps without supervision, and show that it predicts the effect of every supervision scheme we test, so where supervision is placed matters more than how much is provided (§ 5).
- We show that the same effect holds for pretrained language models on English tasks, real programs, and natural language reasoning problems (§ 6).

2 Related Work

Sample complexity and task difficulty. Theoretical work predicts that harder tasks need more samples, with sample complexity growing in the leap complexity of the target [12, 25] and in the number of composed functions [13]. Empirical works have instead mostly studied generalization with the task held fixed, connecting its onset to model capacity [5], optimization [7–9], or the number of distinct training tasks [10, 11]. For example, Morris et al. [5] find that generalization begins once a model's capacity is full, which ties its onset to the model rather than the task and suggests that easy and hard tasks would begin to generalize after a similar amount of data.

Among the few studies that vary task difficulty directly, Yao et al. [26] find that the data needed for implicit multi-hop reasoning grows exponentially in the number of hops, but they measure only two to four hops and only without intermediate steps, so their setting cannot separate the cost of the task from the cost of the computation left unobserved. Chang et al. [27] derive a data scaling law in the size of the token set for a fixed task structure, without characterizing how it grows with the number of hops. We vary algorithmic depth systematically, fit a trend that predicts the sample complexity of deeper tasks, separate depth from the number of possible compositions and from model size, and show that the same trend predicts how intermediate supervision changes sample complexity.

Intermediate Step Supervision. Intermediate steps are widely used in CoT prompting [19], rationale training [28], and process supervision [20, 21], and supervising them can make a task learnable when end-to-end training fails [17, 29–31]. Theory shows that supervising intermediate steps lowers sample complexity, and that supervising every step can remove its dependence on the length of the chain [32–35], and empirically, CoT supervision makes the data required for two-, three-, and five-hop tasks grow at nearly the same rate [27]. Implicit and latent reasoning methods instead remove some or all intermediate steps [36, 37]. Most of these works compare supervising every step with supervising only the final answer, while we keep the algorithm fixed and vary which steps are supervised. We find that sample complexity is set by the unobserved algorithmic depth, so where supervision is placed matters more than how much is provided, and removing intermediate steps has a data cost that grows with the longest run of steps it leaves unobserved.

3 Experimental Setup

We study sample complexity in a compositional setting, where a task applies a sequence of functions, one inside the other, to an input. Every input, intermediate result, and answer is an integer from 0 to 30, and each of our operations ($\mathrm{B} = 8$) maps these 31 values to themselves through a random one-to-one mapping f_i, sampled once and then kept fixed. A composition of algorithmic depth D computes $y = f_{i_\mathrm{D}}(\cdots f_{i_1}(x))$, so there are $\mathrm{N} = \mathrm{B}^\mathrm{D}$ possible compositions. For example, the model sees "$f_3\ f_1\ f_7\ 7\ =$" and must output 22, reached through 19 and 4 (Figure 1a). Because each operation is random, the model cannot know what an operation does until it has seen that operation in training, so it must learn both the operations and how to compose them. Following prior work, we use a prime number of values [7, 8], which keeps affine maps invertible when we change the operation type in Section 4 and makes chance accuracy only $1/31$.

Our training set consists of K distinct compositions, each shown on all 31 inputs. We evaluate on 100 compositions that never appear in training but are built from the same operations, so the model can only succeed by applying the operations it has learned. At $\mathrm{D} = 2$ only 64 compositions exist, so we hold out 10. We count data in distinct compositions rather than examples, since showing the same compositions more often does not help while adding new compositions does. Our setup follows a standard line of controlled studies in which small transformers are trained from scratch on algorithmic or compositional tasks [7, 8, 26, 27, 38]. To supervise intermediate steps, we choose a set of steps ($\mathrm{S} \subseteq \{1, \ldots, \mathrm{D}\}$) whose outputs appear in the target, always including the final step D. For the example above, outcome supervision ($\mathrm{S} = \{3\}$) trains the model to output "22", and full supervision ($\mathrm{S} = \{1, 2, 3\}$) trains it to output "19 4 22". Outputs of steps outside S are removed from the target rather than masked. At test time, the model sees only "$f_3\ f_1\ f_7\ 7\ =$", generates every step in S itself, and is scored only on the final answer.

A run generalizes if its held-out accuracy exceeds 0.8 on unseen data [27]. Whether a run generalizes at a given K also depends on its seed [39], so we train at least three seeds at each of several

values of K around the transition and fit a logistic curve to the fraction of runs that generalize as a function of $\log K$. We define sample complexity (K_{50}) as the value of K at which this curve crosses one half, the standard midpoint of such a curve and the point where the curve is estimated most precisely. Our conclusions do not change for accuracy thresholds of 0.5 or 0.9 (Appendix A.6). We train decoder-only transformers from scratch with two layers, four attention heads, learned positional embeddings, and pre-normalization, with width 192 for the depth experiments and 256 for the supervision experiments. Following Nanda et al. [8], we use AdamW with a learning rate of 10^{-3} and weight decay of 1.0 for up to 100,000 steps.

4 Sample Complexity Depends on Algorithmic Complexity

We first test our central hypothesis, that algorithmic complexity predicts sample complexity. We measure algorithmic complexity with algorithmic depth (D), the number of sequentially dependent operations. Depth is a long-standing measure of computational complexity [14–16], matches how multi-step reasoning problems are made harder by adding dependent steps [40, 41], and can be set exactly in our tasks. Throughout this section, only the final answer is supervised, so the model must learn the whole algorithm from the input and the answer alone.

4.1 Sample complexity grows exponentially with algorithmic depth

We start with permutation operations and increase the algorithmic depth D from two to five. Each additional step multiplies sample complexity by about six, and on a log scale the measurements lie on a straight line (Figure 2, left). Three additional steps raise sample complexity from about 50 to more than 12,000 training compositions, a more than 200-fold increase for a task that is only slightly longer to describe. The growth has a simple source. Every added step multiplies the number of possible compositions the model must learn to handle, $N = B^D$, by the number of operations. Fitting depths two to five gives the scaling trend $K_{50} \approx 1.23\,N^{0.88}$. The exponent is below one, so the model needs a shrinking fraction of all possible compositions as depth grows, whereas a model that learned each composition separately would need a fixed fraction (Appendix B.2). The gap reflects what the model learns once about each operation and reuses across compositions.

The trend predicts tasks it was not fitted on. Before training any depth-six model, we used the trend to predict its sample complexity, and the measurement falls within the confidence interval of the prediction (Figure 2, left, and Appendix B.3). Depth is also not the only way to make this task harder. The number of operations is another way to change the complexity, which widens the choice at each step instead of adding steps. At depth two, increasing the number of operations from 8 to 32 raises N from 64 to 1,024, and the trend fitted on depth alone predicts every measurement along this axis within 30% without refitting (Figure 2, right). Whichever measurable way the complexity of the task is increased, sample complexity follows the same trend in N.

The number of possible compositions does not determine sample complexity. A plausible alternative explanation for these results is that sample complexity depends only on how much the model must learn, here the number of possible compositions N, regardless of how the computation is organized. Three experiments contradict this hypothesis. First, composing the same operations in parallel, by summing their outputs instead of chaining them, keeps N, the operations, and the training data identical, yet prevents generalization at budgets where the chained task generalizes (Appendix B.7). Second, holding N fixed while changing operations for depth changes sample complexity by up to about twofold (Appendix B.6). Third, intermediate supervision leaves the task and N unchanged, yet reduces sample complexity by orders of magnitude, because it changes how much of the computation the model must learn without supervision (more details in Section 5).

4.2 Sample complexity across different compositional tasks

We also run the depth sweep on compositional tasks with other operation types [7, 27, 42, 43] (Figure 2, middle). Permutations are random one-to-one mappings of the values. Affine operations map x to $ax + b \bmod 31$, so a composition of affine operations is again affine, and many different compositions compute the same function. Scramble operations apply an affine map followed by a fixed random permutation, which keeps each operation close to affine but prevents compositions from collapsing onto the same function. Lookup operations are arbitrary mappings that are generally

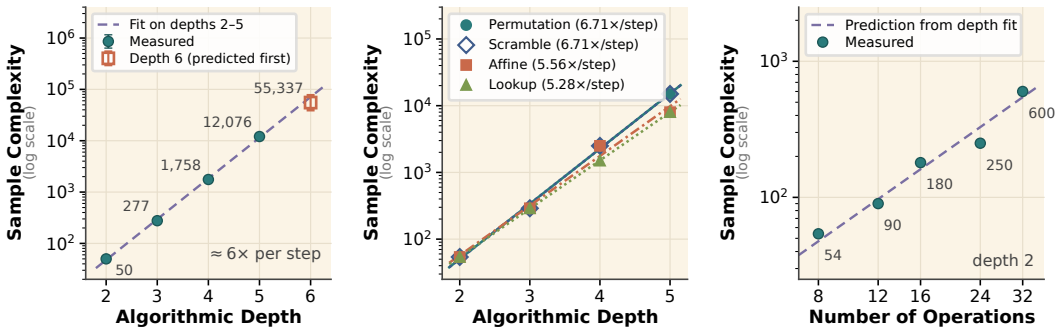

Figure 2: **Sample complexity grows exponentially with algorithmic depth. Left.** Permutation operations, with the scaling trend fitted on depths two to five. The depth-six value was predicted before training. **Middle.** The same growth across four kinds of operations, measured on one shared, coarser set of values of K. **Right.** At depth two, the trend from the left panel predicts sample complexity as the number of operations grows from 8 to 32, without refitting.

not one-to-one, so repeated composition can shrink the set of possible outputs. To compare the four types directly, we measure them on one shared, coarser set of values of K (Appendix B.4).

Every operation type grows exponentially with depth, at its own rate. Sample complexity grows by $6.71\times$ per step for permutation and scramble operations, $5.56\times$ for affine operations, and $5.28\times$ for lookup operations. We read this rate as a characteristic of the task, set by the operations being composed. It is lower for affine and lookup operations, where composition merges many compositions into the same function or shrinks the set of possible outputs, so the model has less to tell apart. The form of the growth, however, is the same for all four. Whenever algorithmic depth increases in a controlled way, sample complexity grows exponentially, at a rate the task determines.

4.3 Sample complexity is largely independent of model size and optimization

We also measure how sample complexity changes with model size, at a fixed algorithmic depth. If generalization began only once a model's memorization capacity was filled [5], larger models would need more data before they generalize. We find a different picture. Model size decides whether the model can represent the algorithm to be learned at all. A one-layer transformer fits its training compositions perfectly but never generalizes in our experiments, however much data it sees, while a two-layer model of the same width learns the task. Once a model can represent the algorithm, its size no longer matters. Models with more than twenty times as many parameters need about the same number of training compositions, with no trend across sizes (Appendix B.8).

Optimization plays a similar role. Work on grokking shows that optimization choices strongly affect when a model generalizes during training [7–9]. Sample complexity asks a different question, how much data a model needs rather than how long it trains. Changing the learning rate over a tenfold range, or switching between the two learning-rate schedules we use, moves sample complexity far less than a single step of depth (Appendix B.9). Model size and optimization decide whether and how quickly a model learns the task, while the task difficulty decides how much data it needs.

Finding 1: ***Sample Complexity Grows Exponentially with Algorithmic Complexity***

Harder tasks need more data to generalize. Sample complexity grows exponentially with algorithmic depth, at a rate set by the task. The trend fitted on shallow tasks predicts the sample complexity of deeper ones before they are trained.

5 Unobserved Algorithmic Depth Predicts the Effect of Supervision

Section 4 measured sample complexity when only the final answer is supervised. In that setting the model must learn the whole algorithm from its input and output alone, so algorithmic depth describes both the task and the learning problem. We now keep the algorithm fixed and change only

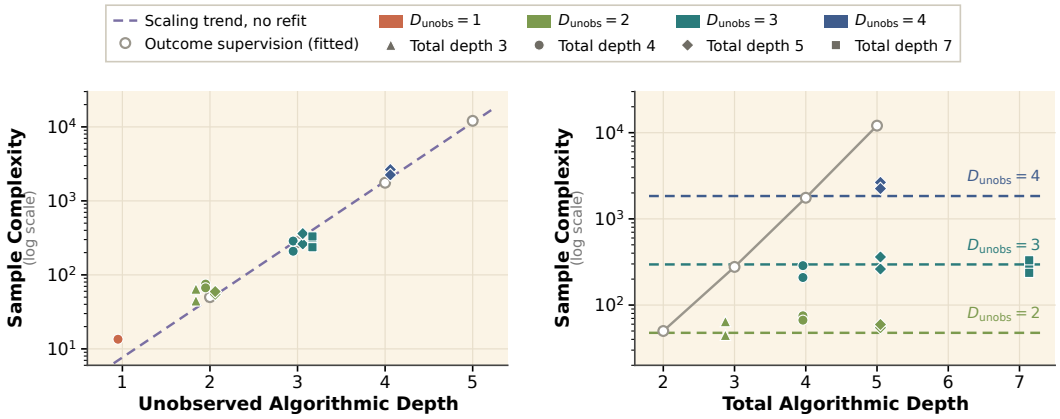

Figure 3: **Unobserved algorithmic depth sets sample complexity. Left.** Every supervision scheme across total depths three to seven, plotted against its unobserved algorithmic depth. All schemes follow the scaling trend of Section 4 without refitting. **Right.** The same schemes plotted against total depth. Under outcome supervision, sample complexity grows steeply with depth. Once the unobserved depth is fixed, sample complexity stays nearly flat as the task gets longer.

which intermediate steps are supervised. This separates the complexity of the algorithm from the complexity of what the model has to learn without guidance.

5.1 Supervision sets the unobserved algorithmic depth

Supervising intermediate steps follows the same idea as CoT training. The training data contains intermediate results of the computation along with the answer, while at test time the model receives only the input, must produce these results itself, and is evaluated only on its final answer. For the same algorithm, supervising intermediate steps reduces sample complexity by orders of magnitude (Figure 1b). Each supervised step gives the model a target partway through the algorithm, which breaks one long computation into shorter pieces that the model learns one after another.

Sample complexity is set by the longest of these pieces. Every piece composes operations from the same set, so once the model learns the longest piece, shorter ones need little additional data. Let $S \subseteq \{1, \ldots, D\}$ be the supervised steps, which always contain the final step D, and count the input as step zero. We define the unobserved algorithmic depth (D_{unobs}) as the largest gap between consecutive steps in $S \cup \{0\}$, the longest run of steps the model must compute without being shown where it ends. Under outcome supervision $D_{unobs} = D$, and under full supervision $D_{unobs} = 1$. On a four-step task, $S = \{2, 4\}$ gives $D_{unobs} = 2$, while $S = \{1, 4\}$ gives $D_{unobs} = 3$.

The scaling trend of Section 4 then predicts the effect of supervision without refitting. The hardest piece has $B^{D_{unobs}}$ possible compositions, and evaluating $K_{50} \approx 1.23\, N^{0.88}$ at $N = B^{D_{unobs}}$ instead of B^D gives the sample complexity of every supervision scheme we test, across total depths from three to seven (Figure 3, left, and Appendix C.1). Schemes with very different numbers of supervised steps and total depths fall on the same trend as long as they leave the same longest unobserved run. Changing the number of operations shifts sample complexity by the amount the trend predicts, so the relation is not specific to one set of operations (Appendix C.4).

5.2 Where supervision is placed matters more than how much is provided

If sample complexity depends on the longest unobserved run, then the position of each supervised step should matter more than the number of supervised steps, and this is what we observe. Two schemes that supervise the same number of steps can differ several-fold in sample complexity when one leaves a longer unobserved run than the other, and an additional supervised step that does not shorten the longest run barely changes sample complexity (Figure 1b and Appendix C.2). With m supervised steps on a D-step task, the best placement therefore spaces them evenly, which gives the shortest possible longest run, $\lceil D/m \rceil$.

Test-time computation does not explain the reduction. Generating intermediate steps also gives a model more computation at test-time, and prior work shows that this added computation lets transformers solve problems they otherwise cannot [44–46]. A plausible hypothesis is that the lower sample complexity comes from the same source, since a model that can compute more at test-time might need fewer samples to reach the same accuracy. Our placement results contradict this hypothesis. Schemes that supervise the same number of steps generate the same number of intermediate outputs, and so use the same test-time computation, yet their sample complexity differs several-fold with the longest unobserved run. Adding a supervised step adds test-time computation but barely helps when that run stays the same. Extra computation may still contribute, but the reduction in sample complexity comes mainly from how supervision divides the learning problem.

Supervision changes generalization, not fitting. A second plausible explanation is that intermediate supervision makes training easier, so supervised models simply fit their training data more reliably. Our runs contradict this explanation as well. In our supervision experiments, 499 of 501 runs fit their training compositions, whether they are supervised at every step or only at the final answer, and excluding the two that do not fit leaves every threshold unchanged (Appendix A.4). The schemes therefore differ not in whether the model fits the data it sees, but in how many compositions it needs before what it learns carries over to compositions it has not seen.

Total depth matters only when it is unobserved. Figure 3 (right) plots every scheme against the total depth of its task. Under outcome supervision, sample complexity grows exponentially with depth, as in Section 4. Once the unobserved depth is fixed, it stays nearly flat as the task grows from two to seven steps (Appendix C.3). A seven-step task with an unobserved depth of three costs about as much as a three-step task under outcome supervision.

Where the longest run starts matters. Among schemes matched on total depth, number of supervised steps, and D_{unobs}, a longest run that starts at the input needs somewhat less data than one that starts at a model-generated step, since an earlier error can carry into a later run during autoregressive evaluation. This effect is much smaller than one additional step of unobserved depth (Appendix C.5).

> **Finding 2:** ***Unobserved Algorithmic Depth Sets Sample Complexity***
>
> Supervising intermediate steps lowers sample complexity by shortening the longest run of steps the model must learn without supervision. The exponential trend between sample complexity and algorithmic depth also holds for this unobserved algorithmic depth, so where supervision is placed matters more than how much is provided.

6 Unobserved Depth Beyond Compositional Tasks

Our compositional tasks let us count every possible composition and measure sample complexity exactly. Real reasoning tasks do not allow this, and they differ in one important way. A pretrained language model already knows many of the operations these tasks use, such as arithmetic and the meaning of words, so it does not need to learn each operation from scratch. We therefore do not expect the scaling trend of Section 4 to predict how many training samples a real task needs. What should carry over is the relative effect of supervision. Within a fixed task and model, leaving more of the reasoning unobserved should make generalization substantially more expensive. We test this in three settings that move step by step away from our controlled tasks, from the same tasks written in English, to real programs, to math word problems.

6.1 Unobserved steps remain costly for pretrained models

We first change the representation and the model while keeping the computation fixed. We write each composition as English sentences and fine-tune a pretrained Pythia-70M [22]. The words that name each operation have random meanings in every run, so pretraining cannot supply the answer, and the model still has to learn the operations from its training data. Under outcome supervision, sample complexity stays close to its value for the symbolic tasks, whether the model is trained from scratch or starts from pretrained weights (Appendix D.1). With intermediate supervision, the pattern of Section 5 appears again (Figure 4, left). Each additional step of unobserved depth multiplies sample complexity by four to five, and under full supervision sample complexity stays nearly flat, between 15 and 24 compositions, as the task grows from three to five steps. These experiments use

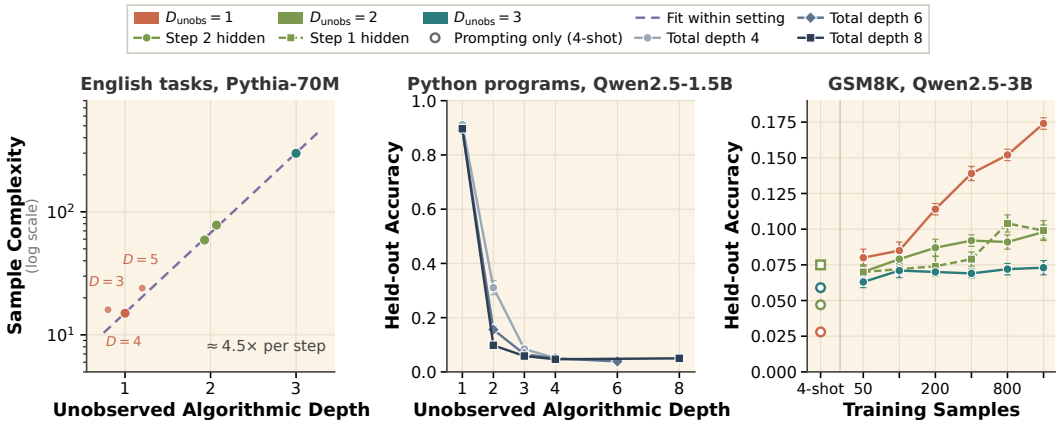

Figure 4: **Unobserved algorithmic depth matters beyond compositional tasks. Left.** The compositional tasks written in English and learned by a pretrained Pythia-70M. Sample complexity grows with unobserved algorithmic depth, and under full supervision it stays nearly flat across total depths three to five. **Middle.** Held-out accuracy on Python programs with 500 training programs. Full supervision works at every program length, while leaving two or more steps unobserved sharply lowers accuracy. **Right.** Held-out accuracy on GSM8K as the number of training samples grows.

fewer runs than the controlled sweeps, so we read them as confirming the direction and rough size of the effect rather than a precise rate.

6.2 Unobserved steps sharply lower accuracy on real programs

We next move to computations that a pretrained model already knows how to perform. We generate Python programs whose steps apply familiar arithmetic, and take every intermediate value directly from the interpreter, following prior work that trains models to predict the results and execution traces of programs [47–49]. The supervised steps are therefore exact, and nothing is annotated by hand. We train on sets of 50 to 8,000 programs and evaluate on 500 held-out programs generated from separate random seeds, so no evaluation program appears in training. A pretrained Qwen2.5-1.5B cannot solve these programs from demonstrations alone, but learns them after fine-tuning. Outcome supervision never reaches a useful accuracy on these programs, so instead of a sample-complexity threshold we compare held-out accuracy at a fixed budget of 500 training programs, where every run fits its training set (Appendix D.3).

With full supervision, the model reaches about 0.9 held-out accuracy at every program length from four to eight steps (Figure 4, middle). Leaving two consecutive steps unobserved lowers accuracy to between 0.1 and 0.3, and leaving three or more leaves the task barely learned at this budget. Real programs also differ from our controlled tasks in one respect. At an unobserved depth of two, longer programs are harder, because they leave more unobserved runs. In the controlled tasks every run composes operations from the same small set, so shorter runs add little once the longest can be learned. In a real program each step is a different computation, so each unobserved run adds its own cost. The longest unobserved run remains the dominant factor, and the number of unobserved runs also matters when the steps differ from each other.

6.3 Unobserved steps raise sample complexity on math word problems

We test natural reasoning problems whose intermediate steps come from human-written solutions. Each GSM8K solution [24] records the value of every intermediate quantity. We parse each solution into a graph of which quantities depend on which, and define the algorithmic depth of a problem as the longest chain of dependent quantities leading to the answer. We use problems of depth three, where full supervision gives $D_{unobs} = 1$, hiding one step gives $D_{unobs} = 2$, and outcome supervision gives $D_{unobs} = 3$. The question is the same under every scheme, and only which values appear in the training target changes. We fully fine-tune Qwen2.5-3B, evaluate autoregressively, and score only the final answer. Every run fits its training set, so differences between schemes reflect data rather

than optimization (Appendix D.2). The model outputs only the supervised values, without free-form reasoning, so that every scheme is scored on the same output format. This keeps absolute accuracies low and not comparable to standard GSM8K results, but fine-tuning improves well beyond the best prompting baseline, so the gains come from learning the computation rather than the format.

With every step supervised, held-out accuracy keeps rising as the number of training samples grows (Figure 4, right). Hiding one step slows this growth. To reach a held-out accuracy of 0.09, the model needs about 110 training problems under full supervision and about 400 with one step hidden, a $3.5\times$ increase with a 95% interval of 2.6–$6.5\times$. With every step hidden, accuracy barely moves above prompting alone, and GSM8K runs out of depth-three problems before outcome supervision reaches the same accuracy. A smaller Qwen2.5-1.5B orders the schemes the same way (Appendix D.2).

Finding 3: ***Unobserved Algorithmic Depth Matters Beyond Compositional Tasks***

For pretrained language models, leaving more intermediate steps unobserved raises sample complexity on English versions of our compositional tasks, on real programs, and on math word problems. Sample complexity grows with unobserved algorithmic depth in every setting.

7 Discussion & Limitations

What does unobserved algorithmic depth imply for reasoning in language models? For language models, our results suggest viewing CoT training and process supervision [19–21] as ways to shorten the unobserved algorithmic depth of a task. A training example with worked intermediate steps can be worth far more than one with only the final answer, and its value depends on how well the steps cover the reasoning. Worked intermediate steps matter most when high-quality data is limited. Training on final answers alone leaves the entire chain of reasoning unobserved, which in our experiments is the most data-hungry regime. Our experiments use supervised training rather than reinforcement learning, but they suggest that learning long chains of new computation from final answers alone should require rapidly more data unless intermediate feedback shortens the unobserved runs. Recent theory characterizes the two extremes of supervising every step or only the answer [32–34], and our experiments show that the range between them is organized by the longest unobserved run.

Does shorter reasoning come at a cost? A growing line of work shortens CoT, by compressing it, skipping steps, or moving it into latent representations, to save tokens at inference time [35–37]. Our results show that saving tokens has a price in training data. Removing intermediate steps lengthens the runs a model must learn without supervision, and sample complexity grows exponentially with the longest such run. The saving in tokens grows linearly with the steps removed, while the cost in data grows exponentially with the unobserved run that compression creates, so how far to compress reasoning depends on how much training data is available.

Limitations. Our most precise measurements use small transformers trained from scratch on controlled tasks, and the language-model evidence rests on fewer runs, with a single measured step on GSM8K. We study supervised training only, not reinforcement learning, and the scaling trend covers chained compositions rather than steps combined in parallel (Appendix B.7). Supervision also brings no benefit when the intermediate facts are already given in the input (Appendix D.4).

8 Conclusion

We asked how much training data a model needs before it generalizes, and found that algorithmic complexity predicts sample complexity. Sample complexity grows exponentially with algorithmic depth, and supervising intermediate steps reduces it by shortening the unobserved algorithmic depth, with the same trend predicting every supervision scheme we test. The same pattern holds for pretrained language models, pointing to a less discussed advantage of chain-of-thought training. Training data has a computational structure as well as a size, and how supervision divides a computation can matter as much as how much data a model sees.

Reproducibility Statement

We provide the information necessary to reproduce our experiments throughout the main paper and appendix. We specify the models and benchmarks used for evaluation, datasets and seeds used for training, and the important training and implementation hyperparameters. Additional experimental and implementation details are provided in the appendix.

AI Use Statement

We used AI tools to assist with writing and editing code and for editing portions of the manuscript. All AI-assisted code and text were reviewed by the authors. The authors take full responsibility for the final content of the paper, including all code, text, claims, and results.

References

[1] Jared Kaplan, Sam McCandlish, Tom Henighan, et al. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[2] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 1

[3] Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Position: Will we run out of data? limits of llm scaling based on human-generated data. In *Forty-first International Conference on Machine Learning*, 2024. 1

[4] Tian Qin, Kimia Hamidieh, and David Alvarez-Melis. Bridging compute-and data-optimal pretraining. *arXiv preprint arXiv:2607.25271*, 2026. 1

[5] John X Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G Edward Suh, Alexander M Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025. 1, 3, 5

[6] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models 3.3: Knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024. 1

[7] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022. 1, 3, 4, 5

[8] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2301.05217. 3, 4

[9] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023. 1, 3, 5

[10] Allan Raventos, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. *arXiv preprint arXiv:2306.15063*, 2023. 1, 3

[11] Florian Redhardt, Yassir Akram, and Simon Schug. Scaling can lead to compositional generalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. Spotlight. arXiv:2507.07207. 1, 3

[12] Emmanuel Abbe, Enric Boix Adsera, and Theodor Misiakiewicz. Sgd learning on neural networks: leap complexity and saddle-to-saddle dynamics. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 2552–2623. PMLR, 2023. 1, 3

[13] Zixuan Wang, Eshaan Nichani, Alberto Bietti, Alex Damian, Daniel Hsu, Jason D Lee, and Denny Wu. Learning compositional functions with transformers from easy-to-hard data. *arXiv preprint arXiv:2505.23683*, 2025. 1, 3

[14] Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM (JACM)*, 14(2):322–336, 1967. 1, 4

[15] Charles H. Bennett. Logical depth and physical complexity. In *The Universal Turing Machine: A Half-Century Survey*, pages 227–257. Oxford University Press, 1988.

[16] Stasys Jukna. Boolean function complexity: advances and frontiers. *Algorithms and combinatorics*, 27, 2012. 1, 4

[17] Çağlar Gülçehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *The Journal of Machine Learning Research*, 17(1):226–257, 2016. 1, 3

[18] Noam Wies, Yoav Levine, and Amnon Shashua. Sub-task decomposition enables learning in sequence to sequence tasks. *arXiv preprint arXiv:2204.02892*, 2022. 1

[19] 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. 1, 3, 9

[20] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022. 3

[21] 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 *International Conference on Learning Representations*, volume 2024, pages 39578–39601, 2024. 1, 3, 9

[22] Stella Biderman et al. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023. 2, 7, 22

[23] Qwen Team. Qwen2 technical report. arxiv. *arXiv preprint arXiv:2412.15115*, 2024. 2, 23, 25

[24] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 2, 8, 23

[25] Emmanuel Abbe, Enric Boix Adsera, and Theodor Misiakiewicz. The merged-staircase property: a necessary and nearly sufficient condition for sgd learning of sparse functions on two-layer neural networks. In *Conference on Learning Theory*, pages 4782–4887. PMLR, 2022. 3

[26] Yuekun Yao, Yupei Du, Dawei Zhu, Michael Hahn, and Alexander Koller. Language models can learn implicit multi-hop reasoning, but only if they have lots of training data. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9695–9713, 2025. 3

[27] Hoyeon Chang, Jinho Park, Hanseul Cho, Sohee Yang, Miyoung Ko, Hyeonbin Hwang, Seungpil Won, Dohaeng Lee, Youbin Ahn, and Minjoon Seo. Characterizing pattern matching and its limits on compositional task structures. *arXiv preprint arXiv:2505.20278*, 2025. 3, 4

[28] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022. 3

[29] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pages 3067–3075. PMLR, 2017. 3

[30] Eran Malach. Auto-regressive next-token predictors are universal learners. *arXiv preprint arXiv:2309.06979*, 2023.

[31] Juno Kim and Taiji Suzuki. Transformers provably solve parity efficiently with chain of thought. In *International Conference on Learning Representations*, volume 2025, pages 44642–44668, 2025. 3

[32] Awni Altabaa, Omar Montasser, and John Lafferty. Cot information: Improved sample complexity under chain-of-thought supervision. *arXiv preprint arXiv:2505.15927*, 2025. 3, 9

[33] Steve Hanneke, Idan Mehalel, and Shay Moran. Sample complexity of autoregressive reasoning: Chain-of-thought vs. end-to-end. *arXiv preprint arXiv:2604.12013*, 2026.

[34] Zhiyuan Li. The optimal sample complexity of learning autoregressive chain-of-thought. *arXiv preprint arXiv:2607.07423*, 2026. 9

[35] Juncai Li, Ru Li, Yuxiang Zhou, Boxiang Ma, and Jeff Z. Pan. Chain of thought compression: A theoretical analysis. *arXiv preprint arXiv:2601.21576*, 2026. 3, 9

[36] Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024. 3

[37] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024. 3, 9

[38] Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024. 3

[39] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020. 3

[40] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018. 4

[41] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 2022. 4, 25

[42] Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2023. 4

[43] William Merrill and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024. 4

[44] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in neural information processing systems*, 36:70757–70798, 2023. 7

[45] Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, volume 2024, pages 11911–11943, 2024.

[46] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. 7

[47] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014. 8

[48] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[49] Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024. 8

A Experimental Setup

A.1 Notation and Terminology

Table 1 collects the notation used throughout the appendix. In the controlled setting, every sample complexity is compared with a single scaling trend, fitted under outcome supervision with $\mathrm{B} = 8$ permutation operations at depths two to five (Table 6):

$$\mathrm{K}_{50} \approx c\,\mathrm{N}^{\alpha}, \qquad c = 1.231, \quad \alpha = 0.879. \tag{1}$$

The two parameters come from a straight-line fit of $\log \mathrm{K}_{50}$ on $\log \mathrm{N}$, and the main text rounds them to 1.23 and 0.88. At a fixed number of operations, each additional step of depth multiplies N by B and therefore multiplies the predicted sample complexity by B^{α}, which we call the *growth per step*. At $\mathrm{B} = 8$, the growth per step is $8^{0.879} \approx 6.22\times$. Every prediction in this appendix evaluates Eq. (1) without refitting either parameter.

Table 1: **Notation.**

Symbol	Meaning
p	Number of symbols. Inputs, intermediate results, and answers lie in $\mathbb{Z}_{\mathrm{p}}$, with $\mathrm{p} = 31$.
B	Number of operations, $\mathrm{B} = 8$ unless stated otherwise.
$f_1, \ldots, f_{\mathrm{B}}$	The operations, each a function from $\mathbb{Z}_{\mathrm{p}}$ to itself.
D	Algorithmic depth, the number of sequentially dependent operations. Called *total depth* when contrasted with $\mathrm{D}_{\text{unobs}}$.
$\mathrm{N} = \mathrm{B}^{\mathrm{D}}$	Number of possible compositions.
K	Training budget, the number of distinct training compositions.
τ	Accuracy threshold above which a run generalizes, $\tau = 0.8$.
K_{50}	Sample complexity, the training budget at which half of the runs generalize (Appendix A.4).
c, α	Prefactor and exponent of the scaling trend, Eq. (1).
S	Supervision scheme, the set of supervised steps. Always contains the final step D.
$\mathrm{D}_{\text{unobs}}$	Unobserved algorithmic depth, the length of the longest unobserved run.
n	Number of training problems or programs in the experiments of Appendix D.

Runs. A *run* is one training run at a fixed seed and training budget. A run *generalizes* if its held-out accuracy exceeds the accuracy threshold $\tau = 0.8$, and it *fits its training set* if its training accuracy exceeds 0.99.

Supervision schemes. Under *outcome supervision*, $\mathrm{S} = \{\mathrm{D}\}$, and under *full supervision*, $\mathrm{S} = \{1, \ldots, \mathrm{D}\}$. Write the elements of $\mathrm{S} \cup \{0\}$ in increasing order as $0 = s_0 < s_1 < \cdots < s_m = \mathrm{D}$, with the input counted as step zero. Between two consecutive supervised outputs, the model must compute steps $s_{k-1} + 1$ to s_k on its own. We call this block the k-th *unobserved run*. It has length $s_k - s_{k-1}$ and *starts* at step s_{k-1}. The unobserved algorithmic depth is the length of the longest unobserved run,

$$\mathrm{D}_{\text{unobs}}(\mathrm{S}) = \max_{1 \le k \le m} (s_k - s_{k-1}).$$

Outcome supervision gives $\mathrm{D}_{\text{unobs}} = \mathrm{D}$, and full supervision gives $\mathrm{D}_{\text{unobs}} = 1$. We say that the longest unobserved run *starts at the input* if the first run is the only run of length $\mathrm{D}_{\text{unobs}}$. Otherwise, at least one run of length $\mathrm{D}_{\text{unobs}}$ begins from an output the model generated itself, and we say that the longest run *starts at a generated step* (Appendix C.5).

A.2 Task Construction

All controlled experiments use $\mathrm{p} = 31$ symbols and B operations $f_1, \ldots, f_{\mathrm{B}} \colon \mathbb{Z}_{\mathrm{p}} \to \mathbb{Z}_{\mathrm{p}}$. In the main experiments, each operation is a random permutation of $\mathbb{Z}_{\mathrm{p}}$, sampled once per seed and shared by every composition in that run. Appendix B.4 describes the other operation types. A composition of algorithmic depth D is an ordered tuple of operation indices $(i_1, \ldots, i_{\mathrm{D}})$, which maps an input x to

$$y = f_{i_{\mathrm{D}}}\big(f_{i_{\mathrm{D}-1}}(\cdots f_{i_1}(x)\cdots)\big),$$

so there are $N = B^D$ possible compositions.

For a training budget K, we sample K distinct compositions and include each on all $p = 31$ inputs, which gives 31K training examples. Held-out compositions are disjoint from the training compositions but use the same operations, and we sample them so that every operation also appears in training. We hold out 100 compositions whenever N allows it. At $B = 8$ and $D = 2$, only 64 compositions exist, so we hold out 10, and the largest possible training budget is $K = 54$ (Appendix A.7). A seed fixes the operations, the training and held-out compositions, and the model initialization, so runs with the same seed and K see identical data in every model and optimization comparison.

A.3 Models and Optimization

Table 2 lists the default configuration, and experiments that change a setting state the change. We use two learning-rate schedules. The *constant* schedule holds the learning rate at its peak after warmup and uses early stopping. The *cosine* schedule decays the learning rate to 5% of its peak and runs without early stopping. The constant schedule is used for the depth sweep up to depth five, the sweep over the number of operations, and the operation-type experiments. The cosine schedule is used at depth six and in every experiment run after it, including all supervision experiments and the summed compositions. Appendix B.9 compares the two schedules on matched configurations.

Table 2: **Default configuration for the controlled experiments.**

Setting	Value
Task	
Symbols p	31
Operations B	8
Held-out compositions	100 (10 at $B = 8$, $D = 2$)
Model	
Architecture	Decoder-only transformer
Layers	2
Width	192 (depth experiments), 256 (supervision experiments)
Attention heads	4
MLP expansion ratio	4
Positional embeddings	Learned
Normalization / activation	Pre-normalization / GELU
Optimization	
Optimizer	AdamW, $(\beta_1, \beta_2) = (0.9, 0.98)$
Peak learning rate	10^{-3}
Weight decay	1.0
Warmup	1,000 steps
Schedule	Constant with early stopping, or cosine to 5% of peak
Gradient clipping	1.0
Precision	bfloat16
Maximum steps	100,000 (depth), 150,000 (supervision)
Maximum batch size	32,768 examples
Evaluation and compute	
Evaluation interval	Every 500 or 1,000 steps
Seeds per training budget	At least 3, more near the transition
Hardware	One NVIDIA A100 or A40 GPU per run

A.4 Measuring Sample Complexity

For each training budget K, we train several runs with different seeds and record whether each run generalizes, that is, whether its held-out accuracy exceeds $\tau = 0.8$. As K grows, the fraction of runs that generalize rises from zero to one, and we call the range of budgets over which it rises the *transition*. We fit a logistic curve to this fraction as a function of $\log K$ and define the sample complexity K_{50} as the budget at which the fitted curve equals one half. Confidence intervals are 95% percentile-bootstrap intervals from 2,000 resamples of the runs. Under outcome supervision, held-out accuracy is exact match on the final answer. Under intermediate supervision, evaluation is

autoregressive. The model generates every supervised step from its own previous outputs, and only the final answer is scored.

Runs that do not fit their training set. A run that does not fit its training set is limited by optimization rather than by data, so we record training accuracy for every run. In the supervision experiments, 499 of 501 runs fit their training set, and every sample complexity is identical with or without the remaining two. At larger depths, whether a run fits its training set and whether it generalizes become correlated, so excluding runs that do not fit would keep the runs that were easiest to optimize and bias K_{50} downward. We therefore compute sample complexities for the depth and operation-type experiments over all runs and report training fit as a separate diagnostic. A model configuration that fits its training set but does not generalize at any budget we tried is treated as below the minimum model size, not as a measurement of sample complexity (Appendix B.8).

A.5 Counting Compositions Rather Than Examples

We measure training data in distinct compositions rather than in examples, and Table 3 shows why. We hold the total number of training examples near 8,000 and increase the number of distinct compositions, showing each one on fewer inputs. With 256 or 512 compositions, every run fits its training set but stays at chance on held-out compositions. With 1,024 compositions and the same number of examples, held-out accuracy rises well above chance, although it stays below τ at this budget. Showing the same compositions on more inputs does not substitute for new compositions.

Table 3: **New compositions matter, repeated examples do not.** The total number of training examples is held near 8,000 while the number of distinct compositions increases. Held-out accuracy is listed for each of three seeds. Chance accuracy is $1/31 \approx 0.032$.

Compositions	Inputs each	Total examples	Training accuracy	Held-out accuracy
256	31	7,936	1.000	0.034, 0.032, 0.033
512	16	8,192	1.000	0.033, 0.035, 0.032
1,024	8	8,192	1.000	0.227, 0.260, 0.252

A.6 Robustness to the Accuracy Threshold

Our conclusions do not depend on the choice of τ. For each of three accuracy thresholds, we recompute the sample complexity at every depth from the same run records and refit the scaling trend (Table 4). The exponent and the growth per step barely change. Because the sample complexities are recomputed rather than taken from Table 6, the exponent at $\tau = 0.8$ (0.864) differs slightly from that of Eq. (1) (0.879).

Table 4: **The scaling trend barely changes with the accuracy threshold.**

Accuracy threshold τ	Exponent α	Growth per step at B $= 8$	R^2
0.5	0.861	5.99×	0.9992
0.8	0.864	6.03×	0.9991
0.9	0.867	6.07×	0.9984

A.7 The Finite Set of Compositions at Depth Two

At B $= 8$ and D $= 2$, only 64 compositions exist and 10 are held out, so the training budget cannot exceed K $= 54$. Table 5 shows the observed transition. The upper plateau, where every run generalizes, cannot be reached, but the rise itself is sharp. This ceiling does not create the trend across depths. Fitting only depths three to five still gives a growth per step of about $6.6\times$.

A.8 Scope and Compute

The results in this paper draw on 7,441 completed training runs in 96 experiment groups, 4,506 in the controlled setting and 2,935 in the natural-language and program experiments, together with

Table 5: **Observed transition at depth two.** The logistic fit gives $K_{50} = 50.0$ (95% CI $[45, 51]$).

Training budget K	Runs	Fraction that generalize
35	43	0.00
40	6	0.00
54	47	0.91

124 evaluations of untrained base models. We exclude from every sample complexity the runs that were superseded, right-censored (stopped before their outcome was known), or affected by an implementation fault. Each run uses a single NVIDIA A100 or A40 GPU with a wall-clock limit of 12 to 24 hours. A single script recomputes every sample complexity from the raw run records, following Appendix A.4. We will release the sweep generator, training code, sample-complexity computation, and per-run records with the paper.

B Algorithmic Depth

B.1 Sample Complexity by Depth

Table 6 lists the sample complexities behind Figure 2 (left). Eq. (1) is fitted to depths two to five. We fixed the prediction for depth six from this fit before training any depth-six model (Appendix B.3).

Table 6: **Sample complexity by algorithmic depth** with $B = 8$ permutation operations and outcome supervision. K_{50}/N is the fraction of all possible compositions needed for training.

D	N	Runs	K_{50}	95% CI	K_{50}/N	Role
2	64	217	50	$[45, 51]$	78.1%	Fit
3	512	523	277	$[267, 288]$	54.1%	Fit
4	4,096	440	1,758	$[1{,}658, 1{,}864]$	42.9%	Fit
5	32,768	469	12,076	$[10{,}935, 13{,}379]$	36.9%	Fit
6	262,144	86	55,337	$[39{,}399, 76{,}810]$	21.1%	Predicted: 71,332

B.2 The Exponent and a No-Reuse Baseline

Because $\alpha < 1$, the fraction of all possible compositions that a model needs, K_{50}/N, falls with depth, from 78% at depth two to 21% at depth six (Table 6). To interpret the exponent, we compare it with coverage baselines. Each baseline assumes that a held-out composition can be solved once a certain part of it has appeared in training. For each baseline, we compute exactly, from the combinatorics of random training sets, the budget K at which half of the held-out compositions have that part covered, and we fit its exponent in N in the same way as Eq. (1).

A model that reused nothing, and could solve a composition only after seeing that exact composition in training, would have an exponent of 1.01 (Table 7). Baselines that require only low-order parts, such as single operations or adjacent pairs, have exponents near zero. The measured exponent of 0.88 lies between these extremes. The model reuses structure across compositions, but far less than a low-order coverage account would suggest.

B.3 Depth Six

We fixed the prediction for depth six from the fit on depths two to five before launching any depth-six run. Eq. (1) predicts $K_{50} = 71{,}332$. Over 86 runs, we measure $K_{50} = 55{,}337$ with a 95% CI of $[39{,}399, 76{,}810]$, which contains the prediction.

Depth six required a change of training recipe. Two early sweeps under the constant schedule did not fit their training sets, reaching at most 25% and 85% training accuracy, so they measured optimization rather than data. A short probe showed that the cosine schedule (Appendix A.3) lets depth-six models fit their training sets. The prediction was fixed before this change, and the recipe

Table 7: **Exponents of coverage baselines.** Each row assumes that a held-out composition is solved once the listed part has appeared in training.

Part that must appear in training	Fitted exponent
Each operation, at any position	-0.03
Each adjacent pair	-0.06
Each operation at its position	0.08
Each pair at its positions	0.13
Each length-two subsequence	0.13
Each length-three subsequence	0.33
The exact composition (no reuse)	1.01
Measured, Eq. (1)	0.88

was not. At depths two to five, the two schedules give sample complexities within $1.5\times$ of each other (Appendix B.9).

Depth six is also the one measurement where we cannot restrict the fit to runs that fit their training set. Only 8 of the 86 runs do. None of the seven at $\mathrm{K} = 5{,}000$ generalizes, and the one at $\mathrm{K} = 48{,}000$ does, so among runs that fit, K_{50} is only bracketed between 5,000 and 48,000. We therefore read depth six as corroborating the scaling trend rather than as an independent test of it, and none of the other conclusions of the paper depend on it.

B.4 Operation Types

We compare four operation types in the same task format.

Permutation. Random invertible functions on $\mathbb{Z}_\mathrm{p}$, as in the main experiments.

Affine. Maps $x \mapsto ax + b \bmod \mathrm{p}$ with $a \neq 0$. A composition of affine maps is again affine, so many different compositions compute the same function. At depth four, the 4,096 possible compositions compute only about 921 distinct functions, out of at most $31 \cdot 30 = 930$ affine maps on $\mathbb{Z}_{31}$.

Scramble. An affine map followed by a fixed random permutation. Each operation stays close to affine, but compositions no longer collapse onto the same function.

Lookup. Arbitrary functions on $\mathbb{Z}_\mathrm{p}$, which are generally not invertible, so repeated composition can shrink the set of possible outputs.

To compare the four types directly, we measure all of them on one shared, coarser grid of training budgets (Table 8). These values are comparable with each other but not directly with the finer estimates in Table 6. Every type grows exponentially with depth. The growth per step differs, and it is lowest for affine and lookup operations, where composition maps many compositions to the same function or shrinks the set of possible outputs.

Table 8: **Sample complexity across operation types**, measured on one shared grid of training budgets. Growth per step comes from a straight-line fit of $\log \mathrm{K}_{50}$ on D.

	K_{50}					
Operation type	$\mathrm{D} = 2$	$\mathrm{D} = 3$	$\mathrm{D} = 4$	$\mathrm{D} = 5$	Growth per step	R^2
Permutation	54	290	2,500	15,000	$6.71\times$	0.998
Scramble	54	290	2,500	15,000	$6.71\times$	0.998
Affine	54	290	2,500	8,000	$5.56\times$	0.988
Lookup	54	290	1,500	8,000	$5.28\times$	1.000

B.5 Number of Operations

At depth two, we increase the number of operations B, which raises the number of possible compositions without adding depth. Table 9 compares each measurement with the prediction of Eq. (1) at $N = B^2$. From 8 to 32 operations, every measurement lies within 30% of its prediction. At 64 operations, the measurement falls about $2\times$ below its prediction. This is also the configuration with the lowest sample complexity in the fixed-N comparison of Appendix B.6.

Table 9: **Number of operations at depth two.** Predictions come from Eq. (1) at $N = B^2$.

Operations B	N	Measured K_{50}	Predicted K_{50}	Measured / predicted
8	64	54	48	1.13
12	144	90	97	0.93
16	256	180	161	1.12
24	576	250	329	0.76
32	1,024	600	545	1.10
64	4,096	847	1,842	0.46

B.6 Fixed Number of Possible Compositions

We hold N fixed and change how it is split between the number of operations and the depth (Table 10). At $N = 4{,}096$, two steps over 64 operations need less than half the data of three or four steps, and the confidence intervals for three and four steps overlap. At $N = 256$, the order reverses, and four steps over 4 operations need less data than two steps over 16 operations. Across both values of N, the split changes sample complexity by at most $2.3\times$ in either direction. This is small compared with the growth per step of about $6\times$, which is why we treat Eq. (1) as a function of N, with depth entering through $N = B^D$.

Table 10: **Same number of possible compositions, different split between operations and depth.**

N	Operations B	Depth D	K_{50}	95% CI
4,096	64	2	847	[815, 1,061]
4,096	16	3	1,925	[1,521, 2,365]
4,096	8	4	1,758	[1,658, 1,864]
256	16	2	153	[142, 166]
256	4	4	114	[77, 131]

B.7 Summed Compositions

Eq. (1) is fitted on chained compositions, in which each operation reads the output of the previous one. To test whether the number of possible compositions alone determines sample complexity, we build summed compositions, which match the chained ones in everything except how the answer is assembled. They use the same operations from the same seed, the same $N = B^D$, the same held-out split, and the same inputs, but every operation reads the original input, and the outputs are combined as

$$y = \sum_{j=1}^{D} 2^j f_{i_j}(x) \bmod p.$$

The powers of two make the answer depend on the order of the operations, so distinct tuples give distinct functions. Summed compositions do not generalize at any budget we tried, even though every run fits its training set (Table 11). At depth four and $K = 2{,}000$, about half of all possible compositions, chained compositions generalize on 7 of 8 seeds with held-out accuracy 0.876, while summed compositions reach at most 0.141. At depth five, every budget we tried lies far below the chained K_{50}, so that row shows only that summed compositions are not easier. The number of possible compositions therefore does not determine sample complexity on its own, and every sample complexity in this paper is measured on chained compositions.

Table 11: **Summed compositions do not generalize.** Every run fits its training set. The chained K_{50} at the same N is shown for reference.

D	N	Budgets K tried	Best held-out accuracy (at K)	Chained K_{50}
3	512	20–300	0.116 (300)	277
4	4,096	30–2,000	0.141 (2,000)	1,758
5	32,768	30–500	0.035 (120)	12,076

B.8 Model Size

We vary model size at depth three (Table 12). A one-layer transformer fits its training set but reaches only about 0.17 held-out accuracy, and none of its 18 runs generalizes at any budget we tried. A two-layer model of the same width learns the task. Above this minimum size, increasing the width from 48 to 256, from about 60K to 1.6M parameters, keeps the sample complexity between 253 and 305 training compositions, with no trend across sizes.

Table 12: **Model size decides whether the task can be learned, not how much data it needs.**

Model	Parameters	Result
1 layer	–	Fits its training set, held-out accuracy ≈ 0.17, no run generalizes
2 layers, widths 48–256	$\approx$ 60K–1.6M	K_{50} between 253 and 305, no trend

B.9 Optimization

Learning rate. At depth three, with everything else fixed, a tenfold sweep of the peak learning rate changes the sample complexity by $1.36\times$ (Table 13), far less than the growth per step.

Table 13: **Learning-rate sweep at depth three.**

Peak learning rate	K_{50}
3×10^{-4}	304
1×10^{-3}	278
3×10^{-3}	223

Learning-rate schedule. The depth sweep up to depth five uses the constant schedule, and depth six and the supervision experiments use the cosine schedule (Appendix A.3). We identified 22 configurations, matched on operation type, B, D, K, width, number of layers, and supervision scheme, that were trained under both schedules. The schedule shifts individual sample complexities by at most about $1.5\times$ (Table 14) and changes the fitted exponent by about 8% (Table 15). These runs come from several experiment groups rather than from a dedicated randomized comparison, so they bound the effect of the schedule rather than show that sample complexity is invariant to it.

C Intermediate Supervision

Throughout this section, a supervision scheme is written as the set S of supervised steps, which always contains the final step D, and D_{unobs} is the length of its longest unobserved run (Appendix A.1). Every prediction evaluates Eq. (1) at $N = B^{D_{unobs}}$ with no parameter refitted. At $B = 8$, this gives predicted sample complexities of 7.7, 47.6, 296.1, and 1,841.9 for $D_{unobs} = 1$, 2, 3, and 4. All supervision experiments use width 256 and the cosine schedule.

C.1 All Supervision Schemes at Depth Four

Table 16 lists the six schemes at depth four behind Figure 1. They span a $130\times$ range in sample complexity, and every measurement lies within $1.8\times$ of its prediction.

Table 14: **Sample complexity under the two learning-rate schedules** on matched configurations.

	Constant schedule		Cosine schedule		
D	K_{50}	95% CI	K_{50}	95% CI	Cosine / constant
2	50.0	[45, 51]	52.0	[46, 56]	1.04×
3	275.7	[263, 288]	285.2	[263, 311]	1.03×
4	1,847.5	[1,747, 1,956]	1,596.0	[1,418, 1,766]	0.86×
5	12,845.4	[11,478, 14,549]	8,728.8	[8,004, 10,787]	0.68×

Table 15: **The scaling trend under each learning-rate schedule.**

Fit	Exponent α	Prefactor c	Growth per step at B = 8
Constant schedule	0.892	1.146	6.39×
Cosine schedule	0.822	1.701	5.52×
Eq. (1)	0.879	1.231	6.22×

C.2 Fixed Number of Supervised Steps at Depth Five

At depth five, every scheme in Table 17 supervises exactly three steps, including the final answer, so the schemes differ only in where the supervised steps are placed. The two schemes with $D_{unobs} = 2$ have a geometric-mean sample complexity of 57.0, and the three schemes with $D_{unobs} = 3$ have 324.6, a 5.70× difference from placement alone. Eq. (1) predicts 6.22× for one additional step of unobserved depth. The identical values for $\{1, 2, 5\}$ and $\{1, 4, 5\}$ do not come from duplicated runs. The two schemes have identical success patterns on the same training budgets, so the logistic fit gives the same K_{50} and interval.

C.3 Total Depth at Fixed Unobserved Depth

At a fixed D_{unobs} and B = 8, every scheme has the same prediction, while the total depth D of the task varies (Table 18). At $D_{unobs} = 3$, total depth ranges from three to seven while sample complexity varies by only 1.73×, with no trend. A regression of $\log K_{50}$ on D gives a growth of about 2% per step of total depth, compared with about 6× per step of unobserved depth. At $D_{unobs} = 2$, total depths from two to five give about 6% per step. The block at $D_{unobs} = 4$ covers only two total depths, which is not enough to estimate a trend. Much of the remaining variation within each block comes from where the longest unobserved run starts (Appendix C.5).

C.4 Changing the Number of Operations

We repeat three supervision schemes with B = 12 operations instead of B = 8, keeping each scheme, and therefore D_{unobs}, fixed (Table 19). If sample complexity is set by $N = B^{D_{unobs}}$, Eq. (1) predicts that it grows by a factor of $(12/8)^{\alpha D_{unobs}}$, which is 2.04× at $D_{unobs} = 2$ and 2.91× at $D_{unobs} = 3$. The measured ratio matches the prediction to within 2% for the scheme with $D_{unobs} = 3$ and falls 17–18% short of it for both schemes with $D_{unobs} = 2$.

Table 16: **Six supervision schemes on the same depth-four task.** |S| is the number of supervised steps, including the final answer.

Scheme S	D_{unobs}	|S|	Predicted K_{50}	Measured K_{50}	95% CI	Measured / predicted
{1, 2, 3, 4} (full)	1	4	7.7	13.5	[10, 18]	1.75
{2, 4}	2	2	47.6	75.1	[71, 77]	1.58
{1, 2, 4}	2	3	47.6	66.8	[66, 71]	1.40
{3, 4}	3	2	296.1	208.8	[207, 211]	0.71
{1, 4}	3	2	296.1	286.6	[273, 364]	0.97
{4} (outcome)	4	1	1,841.9	1,758	[1,658, 1,864]	0.95

Table 17: **Three supervised steps at depth five, placed differently.**

Scheme S	D_{unobs}	Runs	Predicted K_{50}	Measured K_{50}	95% CI	Measured / predicted
$\{1,3,5\}$	2	30	47.6	54.6	$[40,55]$	1.15
$\{2,4,5\}$	2	30	47.6	59.5	$[54,64]$	1.25
$\{3,4,5\}$	3	15	296.1	261.8	$[204,359]$	0.88
$\{1,2,5\}$	3	15	296.1	361.4	$[281,468]$	1.22
$\{1,4,5\}$	3	15	296.1	361.4	$[281,468]$	1.22

Table 18: **Total depth barely matters once unobserved depth is fixed.** All rows use $B = 8$.

Total depth D	Scheme S	K_{50}	Measured / predicted
$D_{unobs} = 2$, predicted $K_{50} = 47.6$			
2	$\{2\}$ (outcome)	50.0	1.05
3	$\{1,3\}$	64.5	1.36
3	$\{2,3\}$	44.7	0.94
4	$\{1,2,4\}$	66.8	1.40
4	$\{2,4\}$	75.1	1.58
5	$\{1,3,5\}$	54.6	1.15
5	$\{2,4,5\}$	59.5	1.25
$D_{unobs} = 3$, predicted $K_{50} = 296.1$			
3	$\{3\}$ (outcome)	277	0.94
4	$\{1,4\}$	286.6	0.97
4	$\{3,4\}$	208.8	0.71
5	$\{1,2,5\}$	361.4	1.22
5	$\{1,4,5\}$	361.4	1.22
5	$\{3,4,5\}$	261.8	0.88
7	$\{2,4,7\}$	289.7	0.98
7	$\{3,5,7\}$	236.5	0.80
7	$\{3,6,7\}$	330.9	1.12
$D_{unobs} = 4$, predicted $K_{50} = 1{,}841.9$			
4	$\{4\}$ (outcome)	1,758	0.95
5	$\{1,5\}$	2,651.5	1.44
5	$\{4,5\}$	2,246.6	1.22

Table 19: **Changing the number of operations moves the sample complexity of supervised schemes by about the predicted amount.**

		K_{50}			Ratio, $B = 12$ over $B = 8$	
Scheme S	D_{unobs}	$B = 8$	$B = 12$	Predicted, $B = 12$	Measured	Predicted
$\{1,2,4\}$	2	66.8	112	97	$1.68\times$	$2.04\times$
$\{2,4\}$	2	75.1	128	97	$1.70\times$	$2.04\times$
$\{1,4\}$	3	286.6	848	862	$2.96\times$	$2.91\times$

C.5 Where the Longest Unobserved Run Starts

Among schemes matched on total depth, number of supervised steps, and D_{unobs}, those whose longest unobserved run starts at the input need less data than those whose longest run starts at a generated step (Table 20). Following Appendix A.1, a scheme counts as starting at the input only if its first unobserved run is its only run of length D_{unobs}. For example, $\{3, 6, 7\}$ has two runs of length three, from the input to step 3 and from step 3 to step 6, so it counts as starting at a generated step. Because evaluation is autoregressive, a run that starts at a generated step begins from the model's own output, so an earlier error can carry into it.

The scheme $\{1, 4, 5\}$ separates this explanation from a simpler one based on position. Its longest run, from step 1 to step 4, sits in the middle of the task, early by position, yet it costs the same as $\{1, 2, 5\}$, whose longest run comes last. At $K = 273$, the one training budget shared by every scheme with $D_{unobs} = 3$, schemes whose longest run starts at the input generalize on 8 of 9 runs and the others on 1 of 12 (Fisher exact test, $p = 0.0004$).

At depth three, we recorded predictions before running the experiments. For $\{2, 3\}$, Eq. (1) at $D_{unobs} = 2$ gives 48. For $\{1, 3\}$, we multiplied this prediction by 1.4 for a longest run that starts at a generated step, giving 67. The measurements are 44.7 and 64.5. At depth seven, $\{3, 5, 7\}$, whose longest run starts at the input, needs less data than both $\{2, 4, 7\}$ and $\{3, 6, 7\}$. The latter two do not differ, even though $\{3, 6, 7\}$ has two runs of length three and $\{2, 4, 7\}$ has one, so the number of longest runs is not what matters. At $D_{unobs} = 4$, the ordering holds, but the confidence intervals overlap.

Table 20: **A longest unobserved run that starts at the input is cheaper.** Each row compares schemes matched on total depth, number of supervised steps, and D_{unobs}.

		Starts at the input		Starts at a generated step	
D	D_{unobs}	Scheme S	K_{50}	Scheme S	K_{50}
3	2	$\{2, 3\}$	44.7	$\{1, 3\}$	64.5
4	3	$\{3, 4\}$	208.8	$\{1, 4\}$	286.6
5	3	$\{3, 4, 5\}$	261.8	$\{1, 2, 5\}$	361.4
				$\{1, 4, 5\}$	361.4
7	3	$\{3, 5, 7\}$	236.5	$\{2, 4, 7\}$	289.7
				$\{3, 6, 7\}$	330.9
5	4	$\{4, 5\}$	2,246.6	$\{1, 5\}$	2,651.5

D Natural Language and Program Experiments

This section gives the construction, training details, and full results for the experiments of Section 6. All of them fine-tune pretrained models. On English compositions, we measure K_{50} as in the controlled setting. On GSM8K and programs, held-out accuracy stays far below τ at the budgets available, so we instead report the number of training examples n needed to reach a *target accuracy*.

D.1 English Compositions with Pythia-70M

We render the controlled compositions as English sentences and fine-tune a pretrained Pythia-70M [22]. Each entity is a single new token that is not in the pretrained vocabulary, so accuracy before training is at chance by construction. Each relation word is an existing single token whose meaning, a random permutation of the entities, is regenerated for each seed, so pretraining cannot supply the answer. The runs use $B = 9$ operations, $p = 31$ symbols, and 100 held-out compositions. We fine-tune all parameters with AdamW, $(\beta_1, \beta_2) = (0.9, 0.98)$, a peak learning rate of 3×10^{-4}, weight decay 0.1, 1,000 warmup steps, a cosine schedule, bfloat16 precision, a maximum batch of 512 examples, and at most 100,000 steps. The supervision comparison at depth four uses 50 runs, with five seeds at each budget near the transition and two elsewhere.

Under outcome supervision at depth three, sample complexity stays in a similar range across surface forms, and whether the model is trained from scratch or fine-tuned from pretrained weights (Table 21). Under intermediate supervision at depth four, sample complexity grows with D_{unobs}, as in the controlled setting (Table 22). Raising D_{unobs} from two to three raises sample complexity by 3.8 to $5.1\times$. At a fixed number of supervised steps, moving from $\{2, 4\}$ to $\{1, 4\}$ gives $5.1\times$, and $\{1, 4\}$ fails on all five seeds at $K = 200$ and succeeds on all five at $K = 340$. Under full supervision, $D_{unobs} = 1$ at every depth, and sample complexity stays nearly flat from depth three to five. The two schemes with $D_{unobs} = 2$ are ordered opposite to the controlled setting, with the scheme that supervises more steps needing more data, but the difference is not significant (Fisher exact test, $p = 0.35$).

Table 21: **Surface form and pretraining at depth three** under outcome supervision.

Surface form	Initialization	Approximate K_{50}
Operation indices	From scratch	277
Symbolic multi-hop relations	From scratch	290
Templated English	From scratch	250
Symbolic multi-hop relations	Pretrained Pythia-70M	290
Templated English	Pretrained Pythia-70M	290

Table 22: **Intermediate supervision on English compositions with pretrained Pythia-70M.** Values are approximate because these experiments use fewer runs than the controlled sweeps.

Total depth D	Scheme S	D_{unobs}	Approximate K_{50}
Supervision schemes at depth four			
4	$\{1, 2, 3, 4\}$ (full)	1	15
4	$\{2, 4\}$	2	59
4	$\{1, 2, 4\}$	2	78
4	$\{1, 4\}$	3	299
Full supervision at depths three to five			
3	$\{1, 2, 3\}$	1	16
4	$\{1, 2, 3, 4\}$	1	15
5	$\{1, \ldots, 5\}$	1	24

D.2 GSM8K

Construction. GSM8K [24] solutions carry calculator annotations of the form `<<48/2=24>>`, so every intermediate quantity the author computed is available. We parse each solution into a dependency graph over these quantities and define the algorithmic depth D of a problem as the length of the longest chain of results leading to the answer. For example, a solution that computes three independent quantities and then sums them has four lines but depth two. The training split holds 1,149, 3,020, 1,892, and 627 problems at depths one to four, and the test split holds 181, 507, 331, and 127. We exclude problems in which two steps on the longest chain have the same value, since the model could then bridge a step by copying. The prompt is `Question: {question} Work:`, and the target lists the supervised values on the longest chain, separated by semicolons. As in the controlled setting, values of unsupervised steps are removed from the target rather than masked. The question is identical across schemes, so at a given depth every scheme uses the same problems and differs only in its target.

Training. We fully fine-tune Qwen2.5-3B [23] with a learning rate of 2×10^{-5}, an effective batch size of 16, and a maximum sequence length of 640 tokens, at six training budgets, $n = 50$, 100, 200, 400, 800, and 1,600 problems. We train six seeds per cell, where a cell is one scheme at one budget. Each run trains for 12 epochs or 500 optimizer steps, whichever is longer. Without this floor, a run with 50 problems would take only 38 steps, and training accuracy would track the number of steps rather than the data, in a way that depends on the scheme. With the floor, all 144 depth-three runs reach training accuracy of at least 0.995. We evaluate on 320 held-out depth-three problems and score only the final answer, with a numeric tolerance of 10^{-6}.

Prompting baselines. We evaluate the untrained model with the same values-only output format. A few demonstrations teach the format but not the task (Table 23). For each scheme, we compare fine-tuning against its best prompting baseline.

Table 23: **Prompting baselines on depth-three GSM8K problems.** Few-shot prompting teaches the output format but does not reproduce the gain from fine-tuning.

		Untrained, prompted			Fine-tuned
Scheme S	D_{unobs}	Zero-shot	Two-shot	Four-shot	$n = 1{,}600$
$\{1, 2, 3\}$ (full)	1	0.022	0.028	0.028	0.174
$\{1, 3\}$	2	0.022	0.047	0.047	0.098
$\{2, 3\}$	2	0.022	0.069	0.075	0.099
$\{3\}$ (outcome)	3	0.022	0.025	0.059	0.073

Sample complexity. Table 24 reports held-out accuracy for every scheme and budget at depth three. For each scheme, we find the number of training problems needed to reach a target accuracy by linear interpolation in $\log n$ between the two budgets that bracket it, with confidence intervals from 2,000 bootstrap resamples over seeds (Table 25). We report three nearby target accuracies and read the middle one. At the lowest target, many resamples of the full-supervision curve already exceed the target at the smallest budget, and we clamp their crossing to $n = 50$, which understates the ratio. Outcome supervision reaches none of the targets within 1,600 problems, so it gives only a lower bound rather than a second ratio. The corpus holds only 1,892 depth-three training problems, so this limit comes from the data, not from compute.

Qwen2.5-1.5B orders the schemes the same way, but many of its runs do not fit their training set, so we read no sample complexity from it. We also do not use problems of depth four. Their 627 training problems and 127 test problems leave all schemes within a narrow band, and two of the schemes do not exceed their prompting baselines after fine-tuning.

Table 24: **Held-out accuracy on depth-three GSM8K problems** with Qwen2.5-3B. Each cell is the mean over six seeds, with the standard error of the last digit in parentheses.

		Training problems n					
Scheme S	D_{unobs}	50	100	200	400	800	1,600
$\{1, 2, 3\}$ (full)	1	0.080(6)	0.085(6)	0.114(4)	0.139(5)	0.152(4)	0.174(4)
$\{1, 3\}$	2	0.070(5)	0.079(2)	0.087(6)	0.092(4)	0.091(5)	0.098(5)
$\{2, 3\}$	2	0.070(5)	0.072(3)	0.074(7)	0.079(5)	0.104(6)	0.099(7)
$\{3\}$ (outcome)	3	0.063(4)	0.071(5)	0.070(2)	0.069(3)	0.072(4)	0.073(5)

Table 25: **Training problems needed to reach each target accuracy on GSM8K.** Outcome supervision does not reach any target within 1,600 problems.

		Target accuracy		
Scheme S	D_{unobs}	0.085	0.090	0.095
$\{1, 2, 3\}$ (full)	1	100	113	128
$\{1, 3\}$	2	170	299	1,213
$\{2, 3\}$	2	472	544	626
$\{3\}$ (outcome)	3	$> 1{,}600$	$> 1{,}600$	$> 1{,}600$
Ratio, $D_{unobs} = 2$ over $D_{unobs} = 1$		$3.1\times$	$3.5\times$	$5.2\times$
95% CI		$[1.9, 6.3]$	$[2.6, 6.5]$	$[2.9, 8.7]$

D.3 Execution Traces

Construction. We generate Python functions whose longest chain of statements into the return value has exactly D statements, where statement k reads the result of statement $k - 1$. The initial value is drawn uniformly from $[20, 400]$, and each step applies one of seven operations, `v+c`,

`v-c`, `v*2`, `v*3`, `v//2`, `v+c*10`, and `v-c*10`, with a constant c drawn from $[2, 9]$. We reject a step if its result has more than three digits or equals its input, and we discard programs in which two intermediate values coincide. Intermediate values come directly from the interpreter through `sys.settrace`. The prompt is the full function source followed by `Work:`, and the target lists the supervised values separated by semicolons, with unsupervised values removed. Training and evaluation programs come from disjoint generator seeds.

Training. We fully fine-tune Qwen2.5-1.5B [23] with AdamW, $(\beta_1, \beta_2) = (0.9, 0.95)$, a learning rate of 2×10^{-5}, a cosine schedule with 5% warmup, and 12 epochs or 500 steps, whichever is longer. We train three seeds per cell on budgets of 50 to 8,000 programs and evaluate on 500 held-out programs. With four demonstrations, the untrained model reproduces the output format perfectly but reaches at most 0.024 accuracy, while the fine-tuned model reaches 0.74 with 250 training programs. An earlier version of this pipeline truncated sequences and trained for too few steps, and it did not learn the task even under full supervision. Its 881 runs are excluded from every number in this paper.

Results. Table 26 reports, for depth-four programs, the number of training programs needed to reach each target accuracy, interpolated as for GSM8K. A longest unobserved run that starts at the input helps while data is scarce, and the advantage fades as the budget grows. At the same D_{unobs}, leaving one more step unobserved costs 2.6 to 3.7× more programs at every target. On programs, the number of unobserved steps therefore adds to the cost set by the longest unobserved run.

Table 26: **Training programs needed to reach each target accuracy on depth-four execution traces.** All three schemes have $D_{unobs} = 2$. The longest run of $\{2, 3, 4\}$ starts at the input and that of $\{1, 2, 4\}$ at a generated step, and $\{2, 4\}$ leaves one more step unobserved than $\{1, 2, 4\}$. A dash means that $\{2, 4\}$ does not reach the target within the budgets tried.

	Programs needed		Ratio	
Target accuracy	$\{2, 3, 4\}$	$\{1, 2, 4\}$	$\{1, 2, 4\}$ over $\{2, 3, 4\}$	$\{2, 4\}$ over $\{1, 2, 4\}$
0.15	58	104	1.79×	2.97×
0.20	106	163	1.54×	2.90×
0.25	142	252	1.77×	2.65×
0.30	219	322	1.47×	2.90×
0.40	478	533	1.11×	3.70×
0.50	895	961	1.07×	3.60×
0.60	1,781	2,117	1.19×	2.67×
0.70	4,377	4,377	1.00×	–

D.4 MuSiQue

We apply the same construction to MuSiQue [41] with Qwen2.5-1.5B, taking the longest chain in the question decomposition as the algorithmic depth. Unlike GSM8K, each question comes with a context. We keep only the paragraphs that the dataset marks as supporting the answer, so a context holds a median of two paragraphs and 186 words, and the first bridge entity, the answer to the first sub-question, appears in the opening paragraph for 94% of training questions. Intermediate supervision does not help in this setting. At depth three, outcome supervision reaches 0.575 held-out accuracy against 0.535 under full supervision. At depth four, outcome supervision reaches 0.512 against 0.428 and 0.460 for the two schemes with $D_{unobs} = 2$. Of the 243 runs, 14 do not fit their training set. All 14 use full supervision at small budgets, so they can only lower the accuracy of full supervision at those budgets.

When the intermediate facts are already given in a short context, an unobserved step does not need to be computed and barely needs to be located. The supervised bridge entities are then extra outputs with no compensating benefit. We read this as a boundary condition. Unobserved algorithmic depth describes computation the model must carry out, not information already present in the input. Restoring the distractor paragraphs, which the dataset provides, would test whether the effect returns when the model has to find the intermediate facts itself.