Quiet Feature Learning in Algorithmic Tasks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

We train Transformer-based language models on ten foundational algorithmic tasks and observe pronounced *phase transitions* in their loss curves that deviate from established power-law scaling trends. Over large ranges of compute, the validation loss barely improves, then abruptly decreases. Probing the models' internal representations reveals that *quiet features* are learned prior to any decrease in task loss. These quiet features represent intermediate algorithmic computations that do not by themselves improve the output loss. Ablation experiments demonstrate that individual quiet features are causally necessary for task performance. Our results demonstrate that substantial representational progress can remain hidden beneath an apparently flat loss curve, challenging the prevailing use of cross-entropy as a proxy for learning and motivating richer diagnostics for monitoring model training.

1 Introduction

Understanding how and when large language models acquire new capabilities has become an important question in deep learning. While language models demonstrated remarkable performance across a broad range of tasks, the precise mechanisms driving their improvements remain unknown. Recent discussions of “emergent abilities" – where larger-scale models outperform baselines abruptly, even though smaller-scale counterparts exhibit little improvement – have led to debate over whether such phenomena are genuine or artifacts of measurement [Wei et al., 2022, Ganguli et al., 2022, Schaeffer et al., 2023].

Questions about emergent abilities are closely tied to the observation of scaling laws in model training [Kaplan et al., 2020, Ruan et al., 2024, Henighan et al., 2020, Dubey et al., 2024, OpenAI, 2023]. These scaling laws typically show a smooth, power-law relationship between compute and model performance. However, most empirical demonstrations of these laws derive from heterogeneous data and tasks, leaving open the possibility that “averaging out" many distinct learning behaviors masks more abrupt transitions that occur for individual skills or subtasks.

In order to better understand skill learning in a tractable setting, we focus on ten foundational algorithmic problems spanning various input types. These algorithmic tasks have precisely defined solutions, making it straightforward to identify clear success criteria, isolate the specific features the model must learn, and ensure that improvements cannot be attributed to memorization or partial heuristics. These tasks allow us to investigate fine-grained learning phenomena which might otherwise be obscured by heterogeneous data.

Our key findings include:

1. **Phase transitions occur during learning**: We observe two distinct phases in scaling laws across tasks and input sizes. In the *slow phase*, loss improves minimally or remains flat.

[*] These authors contributed equally to this work.
Code at https://anonymous.4open.science/r/quiet-feature-learning-in-algorithmic-tasks-0778

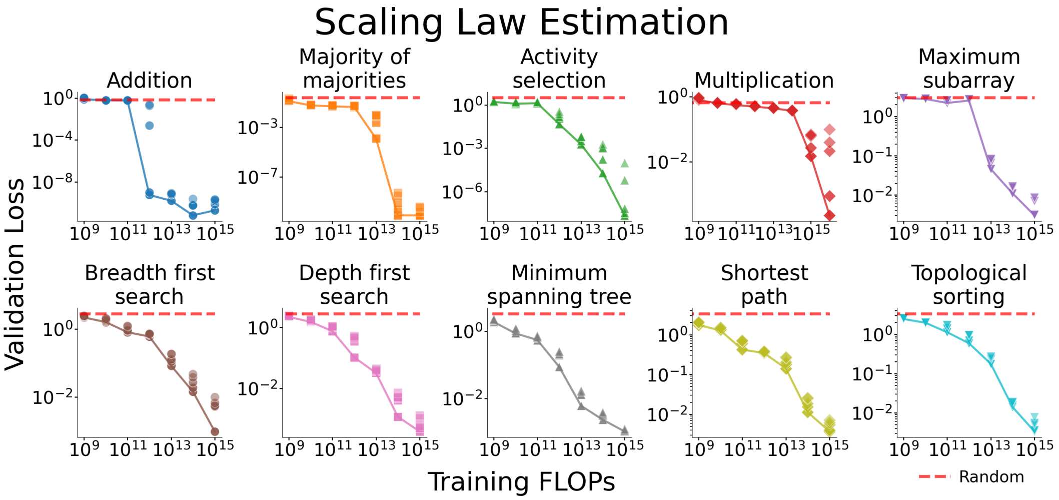

Figure 1: Model performance (validation loss) abruptly improves as we increase the model size, dataset size, and amount of compute (Training FLOPs) used for training. The input size for addition, multiplication, and activity selection is 16. For graph tasks, the input size is 11. For maximum subarray, the input size is 64, while for majority of majorities it is 32. The red dotted line indicates random performance.

Then, loss drops rapidly (*fast phase*). We refer to the change between these two phases as a phase transition. Phase transitions occur for scaling laws estimated across many training runs and within individual training runs.

2. ***Quiet features* precede phase transitions**: Models learn meaningful internal representations during the slow phase, but these features do not yet yield noticeable gains in the output loss (we call these *quiet features*). Ablating them severely degrades performance, demonstrating they are causally related to the eventual sharp drop in loss.

These findings challenge the assumption that improvements in loss necessarily coincide with improvements in feature representations. Instead, substantive internal reorganization may occur below the surface, revealing itself only at discrete points during training.

The rest of this paper is organized as follows: Section 2 reviews related work in scaling laws, emergent abilities, and algorithmic learning. Section 3 describes our experimental methodology and presents our observations of phase transitions across tasks and input sizes. Section 4 introduces our feature analysis framework and demonstrates how quiet and loud features evolve during training. Finally, Section 5 discusses the broader implications of our findings and suggests directions for future research.

2 Related Work

2.1 Scaling Laws

Hestness et al. [2017] observed that scaling dataset size and deep neural network model size led to a predictable decrease in generalization error for neural machine translation, language modeling, image classification and speech recognition. Kaplan et al. [2020] and Hoffmann et al. [2022] observed predictable relationships between training compute and language modeling loss. Henighan et al. [2020] extended this work for generative models across modalities: image, video, multimedia image-text and math. They demonstrated classification loss and error rates predictably decreased on downstream image classification tasks. Chen et al. [2021] studied language model performance on coding. They observed a predictable relationship between language modeling loss on a held out code corpus and model size.

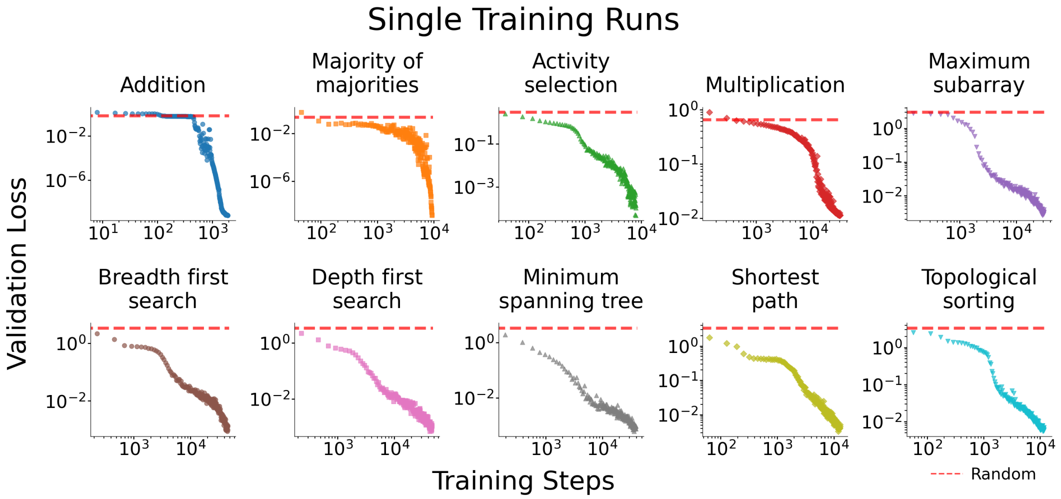

Figure 2: Models exhibit similar abrupt improvement in performance during a single training run. Plots show compute-optimal training runs for the smallest compute budget where test accuracy is 100%. The red dotted line indicates random performance.

2.2 Predicting LLM abilities

Ganguli et al. [2022] and Wei et al. [2022] demonstrated some large language models' capabilities could not be predicted from capabilities of small language models. However, Schaeffer et al. [2023], OpenAI [2023], Ruan et al. [2024], and Dubey et al. [2024] provide evidence that this is due to choice of metrics and that large language models capabilities can be predicted from small language models.

2.3 Proposed explanations for scaling laws

Michaud et al. [2023] proposed that neural networks learn discrete skills called "quanta." They argue that there is a strict ordering, which they called Q sequence, in which quanta must be learned, and that the frequencies of these quanta follow a power law, leading to the power law relationship observed by Kaplan et al. [2020] and others. Hutter [2021] proposes that the relationship between the error rate and dataset size is guided by the distribution of features in the data. They show that a Zipfian distribution of features results in power law scaling.

2.4 Grokking

In grokking [Power et al., 2022, Nanda et al., 2023, Varma et al., 2023], a model trained for many epochs quickly memorizes the training set (thus achieving high training accuracy early) but only later learns a generalizing solution, causing a sudden jump in test accuracy. Our scaling law results are related to grokking, but occur in the single epoch setting. Unlike grokking, models trained in the single-epoch setting do not exhibit a transition from memorization to generalization.

2.5 Progress Measures

Several previous works have identified measures which track progress toward the final, fully-generalizing solution, even when the test loss shows no improvement. Barak et al. [2022] propose a metric for measuring similarity of network weights in the context of sparse parity, and demonstrate that this metric continuously improves throughout training, including prior to measurable improvement in generalization performance. Nanda et al. [2023] propose a different metric on network weights in the context of modular arithmetic grokking, and demonstrate that this metric improves before the phase transition. Mallinar et al. [2024] propose tracking features using Average Gradient Outer Product (AGOP) for Recursive Feature Machines. While this prior work has focused on measuring progress in model weights, they do not demonstrate that the networks are computing interpretable activations prior to generalization. We close this gap by directly probing for human-interpretable features and showing they appear well before the loss drop.

2.6 Phase Transitions

Phase transitions were previously observed for a limited number of algorithmic tasks. Olsson et al. [2022], Garg et al. [2022], and Edelman et al. [2024] find phase transitions for in-context learning during individual training runs. Barak et al. [2022] observed phase transitions in parity. Lee et al. [2024] measure relationships between test accuracy and number of examples (over a fixed model size), with observed phase transitions potentially being explained by the metric artifacts of Schaeffer et al. [2023].

3 Scaling Laws for Algorithmic Tasks

We first aim to estimate scaling laws for 10 foundational algorithmic tasks. Scaling laws are estimated by training models over a range of compute budgets, and identifying the optimal model at each budget.

3.1 Task Formulation

We examine 10 algorithmic tasks which are drawn from three broad categories: binary arithmetic, graph algorithms and sequence-based optimization. The tasks capture a range of input types, and have well-understood algorithms for solving them.

All tasks are formulated as sequence prediction problems. The input to the problem is serialized, and an autoregressive model is trained to predict the solution. All tasks use a standard cross-entropy loss, with the loss masked on the input tokens. We describe how we formulate three of the tasks below. For other tasks, please see Appendix A.

3.1.1 Binary Addition

We formulate n-bit binary addition as the following sequence prediction task:

$$\mathtt{x}_1\mathtt{x}_2\ldots\mathtt{x}_n\mathtt{+}\mathtt{y}_1\mathtt{y}_2\ldots\mathtt{y}_n\mathtt{=}\mathtt{z}_1\mathtt{z}_2\ldots\mathtt{z}_{n+1}\mathtt{<EOS>}$$

where x, y, and z are binary numbers, presented from the least significant bit to the most significant bit. Each bit is represented as a separate token, and +, =, and <EOS> are also represented as individual tokens.

3.1.2 Breadth First Search

Given a connected undirected graph G with n vertices $V = \{v_1, v_2, \ldots, v_n\}$, a set of edges E, and a start vertex v_s, the task is to predict the traversal order in a breadth first search.

We formulate this as:

$$\mathtt{v}_s\mathtt{v}_{i_1}\mathtt{v}_{j_1}\ldots\mathtt{v}_{i_m}\mathtt{v}_{j_m}\mathtt{=}\mathtt{v}_{t_1}\mathtt{v}_{t_2}\ldots\mathtt{v}_{t_n}\mathtt{<EOS>}$$

where $(\mathtt{v}_{i_k}, \mathtt{v}_{j_k})$ represents an edge in E, $m = |E|$ is the number of edges, and $\mathtt{v}_{t_1}\mathtt{v}_{t_2}\ldots\mathtt{v}_{t_n}$ is the complete BFS traversal sequence starting from v_s (where $\mathtt{v}_{t_1} = \mathtt{v}_s$). Ties in BFS ordering are broken by lexicographic ordering.

3.1.3 Maximum Subarray

Given a sequence of n integers $k_1, k_2, \ldots, k_n$ where $k_i \in [-9, 9]$, the maximum subarray task is to predict the contiguous subarray with the maximum sum.

We formulate this as:

$$\mathtt{k}_1\mathtt{k}_2\ldots\mathtt{k}_n\mathtt{=}\mathtt{k}_i\mathtt{k}_{i+1}\ldots\mathtt{k}_j\mathtt{<EOS>}$$

Where $\mathtt{k}_i\mathtt{k}_{i+1}\ldots\mathtt{k}_j$ is the maximum sum subarray ($i \leq j$).

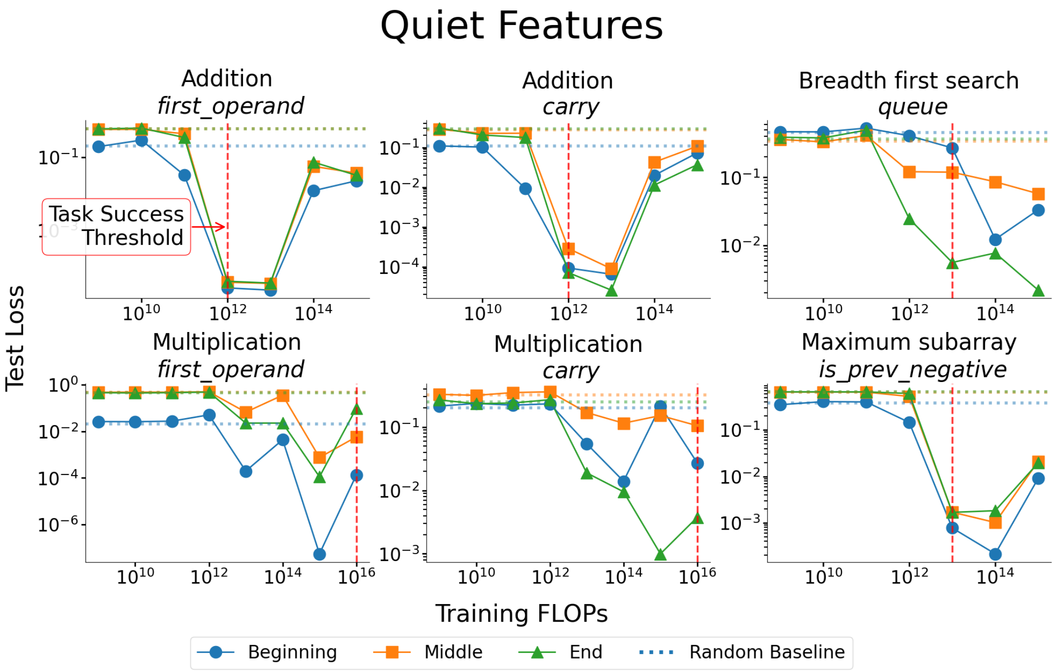

Figure 3: Models learn quiet features before the phase transition. The loss is averaged over the first third of token positions (Beginning), second third (Middle), and last third (End). The red vertical line indicates the task success threshold, which is the smallest compute budget at which the task loss starts to decrease (see Appendix Figure 6). Horizontal dotted lines represent random baselines.

3.2 Experimental Methodology

3.2.1 Model Training

Each task is trained independently with the Transformer++ architecture. Transformer++ [Gu and Dao, 2024] is a decoder-only transformer model with enhancements detailed in Appendix Table 4, based on modifications in Llama and PaLM [Touvron et al., 2023, Chowdhery et al., 2023]. This architecture is chosen because it has improved performance in scaling law experiments compared to other transformer variants [Gu and Dao, 2024]. Models are trained with the AdamW optimizer [Loshchilov and Hutter, 2017] with linear warmup followed with cosine learning rate annealing as prescribed by Hoffmann et al. [2022].

3.2.2 Estimating Scaling Laws

The scaling law experiments aim to estimate the best performance achievable on a task given a compute budget. Separate scaling laws are estimated for each task and input size. Each model is trained up to a pre-specified compute budget, which ranged from $10^9 - 10^{15}$ FLOPs.[1] For each budget, we conduct a grid search across model sizes, batch sizes, and learning rates (see Appendix Table 3 for details about the hyperparameter search). Following the procedure from Chinchilla [Hoffmann et al., 2022], the period of the learning rate scheduler is set to the number of training steps.

The number of training runs per task varies from 1316 to 3565, and the total number of training runs is 18544. All models are trained for at most a single epoch; each algorithmic task has a sufficient number of unique examples to avoid repetition even with the highest compute budgets. The number of training examples is determined based on training compute budget and model size, with all configurations evaluated using randomly generated validation and test sets with 1000 examples each. We choose

[1] For multiplication, the maximum budget was increased to 10^{16}, since this was the minimum budget needed to train the task to 100% accuracy.

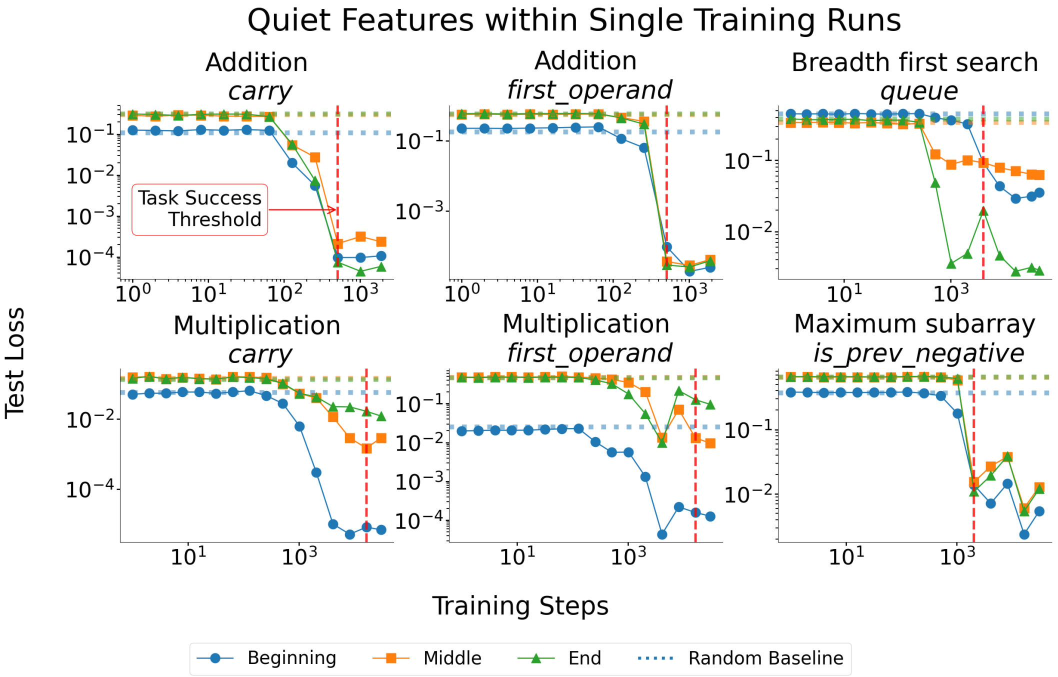

Figure 4: Models learn quiet features before the phase transition within single training runs. The loss is averaged over the first third of token positions (Beginning), second third (Middle), and last third (End). Plots show compute-optimal training runs for the smallest compute budget where test accuracy is 100%. The red vertical line indicates the task success threshold, which is the training step at which the task loss starts to decrease. Horizontal dotted lines represent random baselines.

the configuration with minimum validation loss for each training compute and designate it as the “compute-optimal validation loss."

3.3 Scaling Law Results

We observe phase transitions for compute-optimal validation loss across three scenarios: (1) when we vary both model size & dataset size, (2) when we fix the model size & vary the dataset size, and (3) during individual (compute-optimal) training runs. Figure 1 shows that for six of the tasks, the compute-optimal validation loss undergoes a clear phase transition as the training compute budget increases. For these tasks, there are two distinct phases of learning: a slow phase and a fast phase. During the slow phase, loss is stagnant or decreasing slowly. During the fast phase, the loss decreases rapidly.

For addition, majority of majorities, activity selection and maximum subarray the validation loss is roughly constant in the slow phase then suddenly goes to near zero during the fast phase. For multiplication and breadth first search, the slow phase has a gradual decrease followed by a steeper decrease in the fast phase.

Next, we investigate the effects of varying the dataset size. In Appendix Figure 6, we fix the model size (selecting the model size corresponding to the smallest training compute budget that achieves 100% test accuracy) and increase the dataset size. We continue to observe phase transitions even when the model size is fixed. In this setting, additional graph tasks exhibit distinct phase transitions.

We next analyze model behavior within individual training runs.[2] Figure 2 shows these individual training runs exhibit phase transitions in the loss. For addition and majority of majorities, there is a predictable power-law regime after the phase transition.

[2] These training runs correspond to compute-optimal hyperparameter settings.

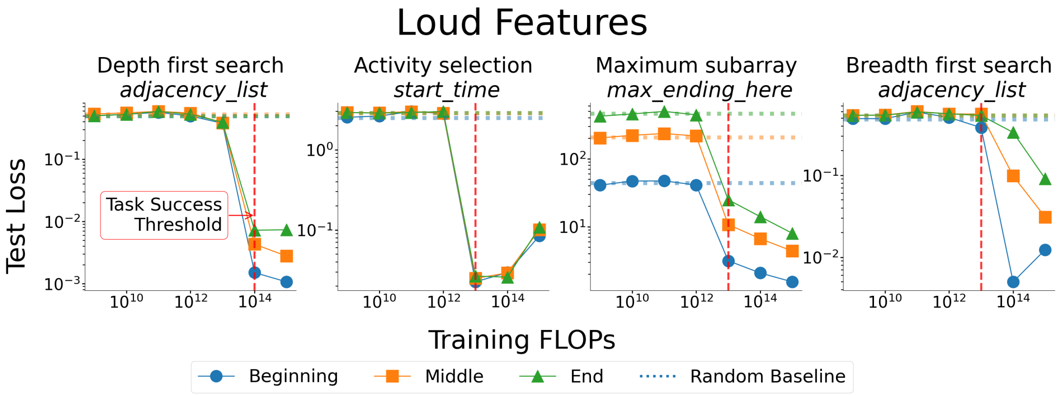

Figure 5: Models learn different set of features (loud features) at or after the phase transition. The loss is averaged over the first third of token positions (Beginning), second third (Middle), and last third (End). The red vertical line indicates the task success threshold, which is the smallest compute budget at which the task loss starts to decrease (see Appendix Figure 6). Cross-entropy loss is used for training probes for *first_operand*, *adjacency_list*. The probing loss is mean squared error for *start_time* and *max_ending_here*, since these are continuous features.

Phase transitions in compute-optimal validation losses occur across different task sizes (see Appendix Figure 7). For addition, phase transitions are observed across task sizes and similarly within individual training runs (Appendix Figure 9). As the input size increases, the Pareto frontier shifts to the right but maintains the same shape. However, for maximum subarray, the phase transition only appears at task sizes greater than 16.

4 Feature Learning before Phase Transitions

In order to better understand the observed phase transitions, we investigate the emergence of human-interpretable features during learning. We focus on features corresponding to intermediate outputs of standard algorithms used to perform the tasks. We use linear probing to identify whether the model learned these features.

4.1 Feature Probing Methodology

For each algorithm-specific feature, we train separate linear probes across each token position and each layer. Probes are trained on the residual streams after each layer (see Appendix B.3). Each probe is trained with 10,000 examples which had been held-out from the original model training set.

For each task, we aim to identify the smallest compute budget at which a feature emerges. We select a single model size to study for this task; models of this size are trained for different compute budgets.[3] We train separate linear probes for each (model, token position, layer) triple. For each (model, token position) pair, we select the probe that achieves the lowest training loss across layers. We report the test performance of the selected probe for each (model, token position) pair.

We establish random baselines by applying the same probing methodology to models initialized with random weights. Test loss is estimated on a separate test set of 1,000 unseen examples.

4.2 Intermediate Task Features

We describe the features investigated for each of the tasks which exhibit phase transitions in their loss. These features are intermediate values computed in standard algorithms for the tasks.

[3] The model size is chosen so that it is nearly optimal across compute budgets. A fixed model size is chosen in order to make feature metrics easier to compare for models across training runs.

Task	Feature	Feature Ablation Δ Accuracy (%)
Addition (16)	*carry*	-41.2^*
Addition (32)	*carry*	-50.4^*
Addition (64)	*carry*	-75.1^*
Addition (16)	*first_operand*	0.00
Addition (32)	*first_operand*	-92.7^*
Addition (64)	*first_operand*	-6.40^*
Multiplication (16)	*carry*	-20.3^*
Multiplication (16)	*first_operand*	-0.05
Maximum Subarray (64)	*is_prev_negative*	-4.14^*
Breadth first search (11)	*queue*	-43.6^*

Table 1: Average difference in test accuracy after ablating a quiet feature compared to ablating a random direction (random ablation). Ablating quiet features degrades test accuracy more than random ablation. For random ablation, we estimate test accuracy over 32 trials. * indicates $p < 0.001$ using bootstrapping. For complete accuracy / loss values see Appendix Table 6

Addition & Multiplication. For n-bit binary addition, we probe for the following at each token $\mathtt{z}_i$: *first_operand*, which is input bit $\mathtt{x}_{i+1}$ (required to compute $\mathtt{z}_{i+1}$); and *carry* c_i, the carry bit used to compute $\mathtt{z}_{i+1}$. Carry c_0 for z_1 is not considered since the first carry is always zero. For multiplication, we check whether the model learns carries generated when adding the last partial product to the sum of the previous $n-1$ partial products.

Breadth/Depth First Search. For breadth first search, we probe at each token $\mathtt{v}_{t_i}$ for the following: *queue*, which is the set of vertices on the queue (in the standard search algorithms) after we have explored vertex $\mathtt{v}_{t_i}$; and *adjacency_list*, which is the set of vertices adjacent to $\mathtt{v}_{t_i}$.

Maximum Subarray. For the maximum subarray problem, we probe at each token $\mathtt{k}_i$ (before the = token) for: *is_prev_negative*, which represents whether $\mathtt{k}_{i-1}$ is negative; and *max_ending_here*, which is the maximum sum of the contiguous subarray ending at $\mathtt{k}_i$. (Refer to Kadane [2023] for the standard algorithm.)

Activity Selection. For the activity selection problem, we probe at each token $\mathtt{f}_i$ for *start_time*, which is the corresponding start time $\mathtt{s}_i$. Since the model has to output $\mathtt{s}_i\mathtt{f}_i$ in order, it must know which start times correspond to which finish times. (Refer to Kleinberg and Tardos [2005] for the standard algorithm.)

4.3 Feature Probing Results

The model learns algorithmic features before, during and after the phase transition. We call features learned prior to the phase transition *quiet features*, as they occur during the slow phase where loss is stagnant or slowly decreasing. Features learned in the fast phase (during and after the phase transition) are *loud features* as the task loss decreases rapidly. Figure 3 shows the trajectory of the probing loss for *quiet features*. For addition, multiplication, and maximum subarray, the model learns features for early token positions prior to the phase transition. However, for breadth first search, later token positions are learned first.

These results apply across distinct, compute-optimal training runs (for a fixed model size). Figure 4 shows that quiet features also emerge during individual training runs. Features for early token positions are also generally learned first in this case.

Figure 5 shows models learn *loud features* in the fast phase (during and after the phase transition).

A surprising finding is the U-shaped feature learning curves in Figure 3, indicating that the probing loss increases for many quiet features after the phase transition. This may indicate that the models are learning alternative representations in the highest compute budget regimes, though the probing loss remains below the random baseline.

4.4 Are *quiet features* causal?

For a given task, we ablate a *quiet feature* from the residual stream at each position, using the feature probes (one probe per position) identified the previous section. We restrict our analyses to binary features. By comparing to ablations of random features, we can evaluate whether a quiet feature is causally responsible for task performance.

We ablate a feature by removing its direction from the residual stream. A linear feature probe $w^\top x^* + b$ outputs 0 (assigns 0.5 probability to each label) when it detects no information from the ablated residual stream at that layer. Letting x be the residual stream at a desired layer, we perform the following optimization:

$$\begin{aligned} \underset{x^*}{\operatorname{argmin}} \quad & \|x - x^*\|^2 \\ \text{subject to} \quad & w^\top x^* + b = 0. \end{aligned}$$

Solving this yields $x^* = x - \frac{w^\top x + b}{||w||^2} w$. The residual stream activation at the linear probe's layer is replaced with x^*.

Ablation results are shown in Table 1. Quiet feature ablations are compared to ablating a random direction. When we ablate quiet features, we observe test accuracy generally degrades more than ablating a random direction, indicating a causal role for quiet features. Similar results are seen for test loss, as shown in Appendix Table 6. However, for *first_operand* at input size 16, we do not see any significant change after ablating the feature compared to random in addition or multiplication. At larger input sizes, ablating *first_operand* for addition leads to significant test loss degradation compared to random ablation.

5 Discussion

Our findings show that, across different algorithmic tasks, there is often a long phase of training with little apparent improvement in next-token prediction loss. Despite this plateau, we observe that essential internal features (e.g., carry bits in binary addition, adjacency in breadth first search) emerge during these periods. These quiet features emerge prior to any substantial improvement in task performance. Ablation experiments confirm that these features are causally important to solving the tasks, suggesting that models can accumulate partial competence that does not immediately translate into lower loss.

One reason for this quiet period may be the all-or-nothing nature of these tasks: obtaining just some of the required subroutines (e.g., some correct carry bits) does not prevent errors on next token prediction. Consequently, any reduction in loss is small until all sub-features are aligned. In over-parameterized models, there is sufficient capacity to learn these subroutines in the background, allowing partial solutions to remain in the representations until they can be combined into a correct overall procedure.

These findings have practical and conceptual implications. For practitioners, they highlight the risk of judging model capabilities based solely on loss curves. Probe or circuit-based diagnostics could provide earlier warnings that a model is nearing a capability threshold. Conceptually, they raise questions about whether similar quiet phases exist in more complex natural-language settings. They also underscore the need for theoretical frameworks that explain why models accumulate latent subroutines before they begin to pay off in observable metrics.

6 Conclusion

We observe Transformer-based models for algorithmic tasks encode important intermediate computations well before they show significant gains in next-token prediction. This *quiet period* exposes a gap between internal representation learning and external task performance, indicating that sub-features may lie dormant until the final pieces align. We hope these insights motivate new methods for probing and monitoring internal learning dynamics – particularly in larger, more complex models – where hidden phases of progress may likewise precede sudden improvements in capability.

References

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham M. Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/884baf65392170763b27c914087bde01-Abstract-Conference.html`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL `https://arxiv.org/abs/2107.03374`.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023. URL `https://jmlr.org/papers/v24/22-1144.html`.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL `https://doi.org/10.48550/arXiv.2407.21783`.

Ezra Edelman, Nikolaos Tsilivis, Benjamin L. Edelman, Eran Malach, and Surbhi Goel. The evolution of statistical induction heads: In-context learning markov chains. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on*

Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/75b0edb869e2cd509d64d0e8ff446bc1-Abstract-Conference.html.

Deep Ganguli, Danny Hernandez, Liane Lovitt, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Nelson Elhage, Sheer El Showk, Stanislav Fort, Zac Hatfield-Dodds, Tom Henighan, Scott Johnston, Andy Jones, Nicholas Joseph, Jackson Kernian, Shauna Kravec, Ben Mann, Neel Nanda, Kamal Ndousse, Catherine Olsson, Daniela Amodei, Tom B. Brown, Jared Kaplan, Sam McCandlish, Christopher Olah, Dario Amodei, and Jack Clark. Predictability and surprise in large generative models. In *FAccT '22: 2022 ACM Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea, June 21 - 24, 2022*, pages 1747–1764. ACM, 2022. doi: 10.1145/3531146.3533229. URL https://doi.org/10.1145/3531146.3533229.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? A case study of simple function classes. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/c529dba08a146ea8d6cf715ae8930cbe-Abstract-Conference.html.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling, 2020. URL https://arxiv.org/abs/2010.14701.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. URL https://arxiv.org/abs/1712.00409.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022. doi: 10.48550/ARXIV.2203.15556. URL https://doi.org/10.48550/arXiv.2203.15556.

Marcus Hutter. Learning curve theory. *CoRR*, abs/2102.04074, 2021. URL https://arxiv.org/abs/2102.04074.

Joseph B. Kadane. Two kadane algorithms for the maximum sum subarray problem. *Algorithms*, 16 (11):519, 2023. doi: 10.3390/A16110519. URL https://doi.org/10.3390/a16110519.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005. ISBN 0321295358.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dsUB4bst9S.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

Neil Mallinar, Daniel Beaglehole, Libin Zhu, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Emergence in non-neural models: grokking modular arithmetic via average gradient outer product. *CoRR*, abs/2407.20199, 2024. doi: 10.48550/ARXIV.2407.20199. URL https://doi.org/10.48550/arXiv.2407.20199.

Brendan D. McKay. Combinatorial data – graphs, 2025. URL https://users.cecs.anu.edu.au/~bdm/data/graphs.html. Accessed: 2025-03-23.

Eric J. Michaud, Ziming Liu, Uzay Girit, and Max Tegmark. The quantization model of neural scaling. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/5b6346a05a537d4cdb2f50323452a9fe-Abstract-Conference.html.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *CoRR*, abs/2201.02177, 2022. URL https://arxiv.org/abs/2201.02177.

Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance. *CoRR*, abs/2405.10938, 2024. doi: 10.48550/ARXIV.2405.10938. URL https://doi.org/10.48550/arXiv.2405.10938.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/adc98a266f45005c403b8311ca7e8bd7-Abstract-Conference.html.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi: 10.48550/ARXIV.2302.13971. URL https://doi.org/10.48550/arXiv.2302.13971.

Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *CoRR*, abs/2309.02390, 2023. doi: 10.48550/ARXIV.2309.02390. URL https://doi.org/10.48550/arXiv.2309.02390.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=yzkSU5zdwD. Survey Certification.

A Task formulation

Binary Addition

Binary addition involves adding two n-bit numbers to produce an $(n+1)$-bit result. We formulate this as a sequence prediction task:

$$\mathtt{x}_1\mathtt{x}_2 \ldots \mathtt{x}_n\mathtt{+}\mathtt{y}_1\mathtt{y}_2 \ldots \mathtt{y}_n\mathtt{=}\mathtt{z}_1\mathtt{z}_2 \ldots \mathtt{z}_{n+1}\mathtt{<EOS>}$$

Where $\mathtt{x}$, $\mathtt{y}$, and $\mathtt{z}$ represent binary numbers, with $\mathtt{x}_1$ denoting the least significant bit (LSB). Each bit is represented as a separate token, and `+`, `=`, and `<EOS>` are special tokens.

Binary Multiplication

Binary multiplication combines two n-bit numbers to produce a $2n$-bit result. We formulate this as:

$$\mathtt{x}_1\mathtt{x}_2 \ldots \mathtt{x}_n\mathtt{*}\mathtt{y}_1\mathtt{y}_2 \ldots \mathtt{y}_n\mathtt{=}\mathtt{z}_1\mathtt{z}_2 \ldots \mathtt{z}_{2n}\mathtt{<EOS>}$$

Following the same convention as in binary addition, with bits ordered from least significant bit to most significant bit.

Majority of Majorities

Given an n-bit number (where n is divisible by 4), we partition the bits into 4 equal consecutive groups. For each group, we compute its majority bit value g_i. The final output is the majority bit value among g_1, g_2, g_3, g_4.

We formulate this as:

$$\mathtt{x}_1\mathtt{x}_2 \ldots \mathtt{x}_n\mathtt{=}\mathtt{z}_1\mathtt{<EOS>}$$

Where $\mathtt{z}_1$ is the final majority bit.

Breadth First Search

Given a connected undirected graph G with n vertices $V = \{v_1, v_2, \ldots, v_n\}$, a set of edges E, and a start vertex v_s, we predict the BFS traversal order.

We formulate this as:

$$\mathtt{v}_s\mathtt{v}_{i_1}\mathtt{v}_{j_1} \ldots \mathtt{v}_{i_m}\mathtt{v}_{j_m}\mathtt{=}\mathtt{v}_{t_1}\mathtt{v}_{t_2} \ldots \mathtt{v}_{t_n}\mathtt{<EOS>}$$

Where $(\mathtt{v}_{i_k}, \mathtt{v}_{j_k})$ represents an edge in E, $m = |E|$ is the number of edges, and $\mathtt{v}_{t_1}\mathtt{v}_{t_2} \ldots \mathtt{v}_{t_n}$ is the complete BFS traversal sequence starting from v_s (where $\mathtt{v}_{t_1} = \mathtt{v}_s$).

Depth First Search

This follows the same formulation as BFS, but the expected output $\mathtt{v}_{t_1}\mathtt{v}_{t_2} \ldots \mathtt{v}_{t_n}$ represents the DFS traversal order:

$$\mathtt{v}_s\mathtt{v}_{i_1}\mathtt{v}_{j_1} \ldots \mathtt{v}_{i_m}\mathtt{v}_{j_m}\mathtt{=}\mathtt{v}_{t_1}\mathtt{v}_{t_2} \ldots \mathtt{v}_{t_n}\mathtt{<EOS>}$$

Shortest Path

Given a connected undirected graph G with n vertices, a set of edges E, and two vertices v_s (source) and v_f (destination), we predict the shortest path between them.

We formulate this as:

$$\mathtt{v}_s\mathtt{v}_f\mathtt{v}_{i_1}\mathtt{v}_{j_1} \ldots \mathtt{v}_{i_m}\mathtt{v}_{j_m}\mathtt{=}\mathtt{v}_{p_1}\mathtt{v}_{p_2} \ldots \mathtt{v}_{p_k}\mathtt{<EOS>}$$

Where $\mathtt{v}_{p_1}\mathtt{v}_{p_2} \ldots \mathtt{v}_{p_k}$ is the shortest path from v_s to v_f (with $\mathtt{v}_{p_1} = \mathtt{v}_s$ and $\mathtt{v}_{p_k} = \mathtt{v}_f$).

Topological Sorting

Given a directed acyclic graph (DAG) G with n vertices and a set of edges E, we predict a valid topological ordering of vertices.

We formulate this as:

$$\mathtt{v}_{i_1}\mathtt{v}_{j_1}\ldots\mathtt{v}_{i_m}\mathtt{v}_{j_m}\mathtt{=}\mathtt{v}_{t_1}\mathtt{v}_{t_2}\ldots\mathtt{v}_{t_n}\mathtt{<EOS>}$$

Where $(\mathtt{v}_{i_k}, \mathtt{v}_{j_k})$ represents a directed edge from $\mathtt{v}_{i_k}$ to $\mathtt{v}_{j_k}$, and $\mathtt{v}_{t_1}\mathtt{v}_{t_2}\ldots\mathtt{v}_{t_n}$ is a valid topological ordering.

Minimum Spanning Tree

Given a connected undirected graph G with n vertices and a set of weighted edges E, we predict the set of edges forming the minimum spanning tree (MST).

We formulate this as:

$$\mathtt{v}_{i_1}\mathtt{v}_{j_1}\mathtt{w}_1\ldots\mathtt{v}_{i_m}\mathtt{v}_{j_m}\mathtt{w}_m\mathtt{=}\mathtt{v}_{p_1}\mathtt{v}_{q_1}\ldots\mathtt{v}_{p_{n-1}}\mathtt{v}_{q_{n-1}}\mathtt{<EOS>}$$

Where $(\mathtt{v}_{i_k}, \mathtt{v}_{j_k}, \mathtt{w}_k)$ represents an edge with weight $\mathtt{w}_k$, and $\{(\mathtt{v}_{p_1}, \mathtt{v}_{q_1}), \ldots, (\mathtt{v}_{p_{n-1}}, \mathtt{v}_{q_{n-1}})\}$ are the edges in the MST.

Maximum Subarray

Given a sequence of n integers $k_1, k_2, \ldots, k_n$ where $k_i \in [-9, 9]$, we predict the contiguous subarray with the maximum sum.

We formulate this as:

$$\mathtt{k}_1\mathtt{k}_2\ldots\mathtt{k}_n\mathtt{=}\mathtt{k}_i\mathtt{k}_{i+1}\ldots\mathtt{k}_j\mathtt{<EOS>}$$

Where $\mathtt{k}_i\mathtt{k}_{i+1}\ldots\mathtt{k}_j$ is the maximum sum subarray ($i \leq j$), and for a single-element result, only $\mathtt{k}_i$ is the output.

Activity Selection

Given a sequence of n activities represented by their start times $(s_1, s_2, \ldots, s_n)$ and finish times $(f_1, f_2, \ldots, f_n)$, we predict the largest subset of non-overlapping activities.

We formulate this as:

$$\mathtt{s}_1\mathtt{s}_2\ldots\mathtt{s}_n\mathtt{f}_1\mathtt{f}_2\ldots\mathtt{f}_n\mathtt{=}\mathtt{s}_{i_1}\mathtt{f}_{i_1}\ldots\mathtt{s}_{i_k}\mathtt{f}_{i_k}\mathtt{<EOS>}$$

Where $\mathtt{s}_{i_1}\mathtt{f}_{i_1}\ldots\mathtt{s}_{i_k}\mathtt{f}_{i_k}$ represents the selected non-overlapping activities in ascending order of finish times.

B Experimental Methodology

B.1 Generating Samples

Binary Tasks. For addition and multiplication, pairs of n-bit binary numbers (a, b) are uniformly sampled without replacement. To prevent memorization, if a pair (a, b) appears in the training set, then (b, a) is removed from the validation and test sets. The input for majority of majorities is a single bit string, which is sampled uniformly without replacement.

Graph Tasks. For graph-based tasks (breadth first search, depth first search, shortest path, minimum spanning tree and topological sorting), we uniformly sample non-isomorphic undirected connected graphs, using the graph dataset of McKay [2025], and randomly permute the vertex labels. For topological sorting, edge directions are determined by randomly sampling a vertex ordering.

Integer Sequence Tasks For maximum subarray and activity selection, we uniformly sample multisets without replacement.

B.2 Estimating Scaling Laws (Additional Details)

During grid search, we filter out hyperparameter combinations that exceed a pre-defined maximum number of steps.[4] We ensure at least one trained model across compute budgets reaches 100% test accuracy on a 1000-example held-out set.

Task	Input Sizes
Addition	8, 16, 32, 64, 128
Multiplication	16, 32
Majority of Majorities	32, 64
Breadth First Search	10, 11
Depth First Search	10, 11
Shortest Path	10, 11
Topological Sorting	10, 11
Minimum Spanning Tree	10, 11
Maximum Subarray	8, 16, 32, 64
Activity Selection	8, 16, 32

Table 2: Computational tasks and their corresponding input sizes used in our experiments.

Hyperparameter	Range
Model Dimension	[8, 16, 32, 64, 128, 256, 512]
Number of Layers	[4, 16]
Number of Heads	4
Batch Sizes	[8, 64]
Peak Learning Rate	$[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}]$
Maximum Steps	10^5 (10^7 for compute $> 10^{15}$ FLOPs)

Table 3: Hyperparameter ranges used in our grid search.

Component	Implementation
Normalization	Pre-Norm, RMSNorm
Positional Embeddings	RoPE
Feed-forward Network	SwiGLU
AdamW betas	0.9, 0.95
Linear Bias	False
Learning Rate Scheduler	Linear Warmup (from 0.01 of peak LR over 10% of training steps) + Cosine Decay to 0.1 of peak LR

Table 4: Architectural modifications used in our Transformer++ implementation.

B.3 Training Feature Probes

B.3.1 Transformer Architecture and Residual Stream

Consider a transformer model with L layers. For each layer $l \in \{1, 2, ..., L\}$ and token position t, we define the layer computation as:

[4] Binary addition, the first task investigated, did not have this restriction on number of steps.

$$\mathbf{x}_{\text{mid}}^{(l,t)} = \mathbf{x}_{\text{pre}}^{(l,t)} + \sum_{\text{head } h} \text{attn}^{(l,h)} \left(\mathbf{x}_{\text{pre}}^{(1,t)}, \mathbf{x}_{\text{pre}}^{(1,1:t)} \right) \tag{1}$$

$$\mathbf{x}_{\text{post}}^{(l,t)} = \mathbf{x}_{\text{mid}}^{(l,t)} + \text{MLP}^{(l)} \left(\mathbf{x}_{\text{mid}}^{(l,t)} \right) \tag{2}$$

where:

- $\mathbf{x}_{\text{pre}}^{(l,t)} \in \mathbb{R}^d$ is input to the layer l at position t (the pre-residual stream). d is the transformer model dimension.
- $\mathbf{x}_{\text{mid}}^{(l,t)} \in \mathbb{R}^d$ is the mid-residual stream (after attention)
- $\mathbf{x}_{\text{post}}^{(l,t)} \in \mathbb{R}^d$ is the output of layer l (post-residual stream)
- $\text{attn}^{(l,h)}$ denotes the h-th attention head in layer l
- $\text{MLP}^{(l)}$ denotes the feedforward network in layer l

We train linear probes on the output of the layer, $\mathbf{x}_{\text{post}}^{(l,t)}$ for each layer l and token position t.

B.3.2 Probe Training Procedure

For each feature f at token position t and layer l, we train a probe $p_{f,l,t}$ on the output of layer l, $\mathbf{x}_{\text{post}}^{(l,t)}$. The type of probe depends on the feature:

Binary Features. For binary feature $f \in \{0, 1\}$, we train a logistic regression classifier:

$$p_{f,l,t}(\mathbf{x}_{post}^{(l,t)}) = \sigma(\mathbf{w}_{f,l,t}^T \mathbf{x}_{post}^{(l,t)} + b_{f,l,t}) \tag{3}$$

where σ is the sigmoid function, $\mathbf{w}_{f,l,t} \in \mathbb{R}^d$, and $b_{f,l,t} \in \mathbb{R}$. The following features are binary: *first_operand*, *carry*, *is_prev_negative*.

Multi-valued Features. Features *queue* & *adjacency_list* represent list of binary variables. For example, *adjacency_list* at token t is a list $(e_1, \ldots, e_k)$ where $e_j \in 0, 1$ represents whether vertex v_t is connected with vertex v_j. To detect such features, we train k independent logistic classifiers:

$$p_{f,l,t}^{(i)}(\mathbf{x}_{post}^{(l,t)}) = \sigma(\mathbf{w}_{f,l,t}^{(i)T} \mathbf{x}_{post}^{(l,t)} + b_{f,l,t}^{(i)}) \quad \text{for } i = 1, ..., k \tag{4}$$

Real-valued Features. For continuous features, *max_ending_here* & *start_time*, we train a linear regressor:

$$p_{f,l,t}(\mathbf{x}) = \mathbf{w}_{f,l,t}^T \mathbf{x} + b_{f,l,t} \tag{5}$$

B.3.3 Training Configuration

All probes are trained using the configuration noted in Table 5.

Parameter	**Value**
Training examples	10,000
Regularization strength (C)	100
Fit intercept	True
Maximum iterations	1,000
Optimizer	L-BFGS (scikit-learn default)

Table 5: Probe training hyperparameters

B.3.4 Probe Selection

Given a trained model with compute budget B, we select the best probe for each feature f and token position t as follows:

$$l^*_{f,t} = \underset{l \in \{1,\ldots,L\}}{\arg\min}\, \mathcal{L}_{\text{train}}(p_{f,l,t}) \tag{6}$$

where $\mathcal{L}_{\text{train}}$ denotes the training loss (cross-entropy for classification, mean squared error for regression). The test performance is then evaluated using probe $p_{f,l^*_{f,t},t}$ on a held-out test set of 1,000 examples.

All our training was done on an 8U HGX server with Dual Intel Sapphire Rapids and 8 NVIDIA H100 GPUs. Test accuracies for feature ablation were computed on a machine with Intel(R) Xeon(R) Gold 6230 CPU and NVIDIA GeForce RTX 2080 Ti.

Task	**Feature**	**Baseline**		**Feature Ablation**		**Random Ablation**	
		Acc. (%)	Loss	Acc. (%)	Loss	Acc. (%)	Loss
Addition (16)	*carry*	100 [99.6, 100]	7.91e-10 [6.59e-10, 9.29e-10]	58.8 [55.8, 61.9]	3.12e-2 [3.06e-02, 3.18e-02]	100 [99.6, 100]	7.96e-10 [7.73e-10, 8.20e-10]
Addition (32)	*carry*	100 [99.6, 100]	1.53e-10 [9.20e-11, 2.42e-10]	49.6 [46.5, 52.7]	2.87e-2 [2.70e-2, 3.04e-2]	100 [99.6, 100]	1.66e-10 [1.497e-10, 1.822e-10]
Addition (64)	*carry*	100 [99.6, 100]	4.68e-10 [3.54e-10, 6.16e-10]	24.9 [22.3, 27.6]	4.60e-2 [4.39e-2, 4.81e-2]	100 [99.6, 100]	1.25e-9 [1.20e-9, 1.32e-9]
Addition (16)	*first_operand*	100 [99.6, 100]	7.91e-10 [6.59e-10, 9.22e-10]	100 [99.62, 100]	7.91e-10 [6.65e-10, 9.29e-10]	100 [99.6, 100]	8.68e-10 [8.44e-10, 8.93e-10]
Addition (32)	*first_operand*	100 [99.6, 100]	1.53e-10 [8.86e-11, 2.42e-10]	7.00 [5.40, 8.60]	7.57e-1 [7.25e-1, 7.89e-1]	99.7 [99.6, 100]	3.46e-4 [2.58e-4, 4.43e-4]
Addition (64)	*first_operand*	100 [99.6, 100]	4.68e-10 [3.51e-10, 6.10e-10]	93.6 [92.1, 95.1]	3.65e-3 [2.72e-3, 4.66e-3]	100 [99.6, 100]	1.07e-8 [6.88e-9, 1.57e-8]
Multiplication (16)	*carry*	76.8 [74.2, 79.3]	1.07e-2 [9.79e-3, 1.16e-2]	56.4 [53.3, 59.5]	5.55e-2 [4.98e-2, 6.17e-2]	76.7 [76.2, 77.2]	1.10e-2 [1.08e-2, 1.17e-2]
Multiplication (16)	*first_operand*	76.8 [74.2, 79.3]	1.07e-2 [9.79e-3, 1.16e-2]	76.8 [74.2, 79.3]	1.07e-2 [9.79e-3, 1.16e-2]	76.9 [76.2, 77.3]	1.07e-2 [1.06e-2, 1.09e-2]
Maximum Subarray (64)	*is_prev_negative*	95.6 [94.3, 96.8]	1.51e-2 [9.25e-3, 2.19e-2]	89.9 [88.0, 91.7]	3.15e-2 [2.32e-2, 4.07e-2]	94.4 [93.8, 94.3]	1.84e-2 [1.72e-2, 1.97e-2]
Breadth first search (11)	*queue*	99.7 [99.1, 99.9]	8.71e-4 [3.23e-4, 1.63e-3]	54.6 [51.5, 57.6]	1.30e-1 [1.20e-1, 1.40e-1]	98.2 [98.1, 98.4]	4.87e-3 [4.44e-3, 5.35e-3]

Table 6: Test accuracy & loss with and without feature ablations on 1000 examples for different tasks and input sizes. Baseline refers to the language model's accuracy without any perturbations. For random ablation, we report the mean over 32 trials. 95% confidence intervals shown in smaller text below each value.

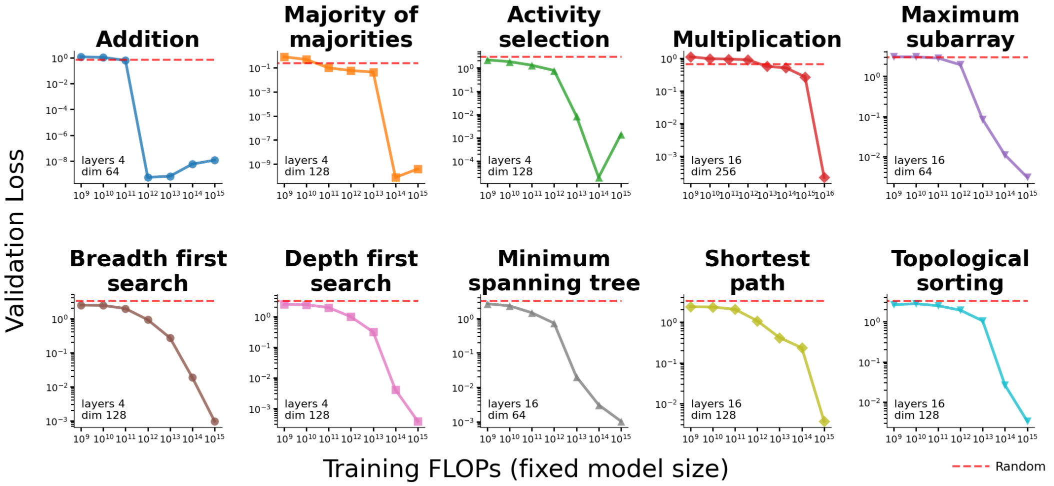

Figure 6: Despite holding model size constant, model performance shows abrupt improvement across various amount of training compute. Plot the minimum validation loss for each compute. Model sizes are compute-optimal for the earliest training compute where test accuracy is 100%. The input sizes are the same as in Figure 1.

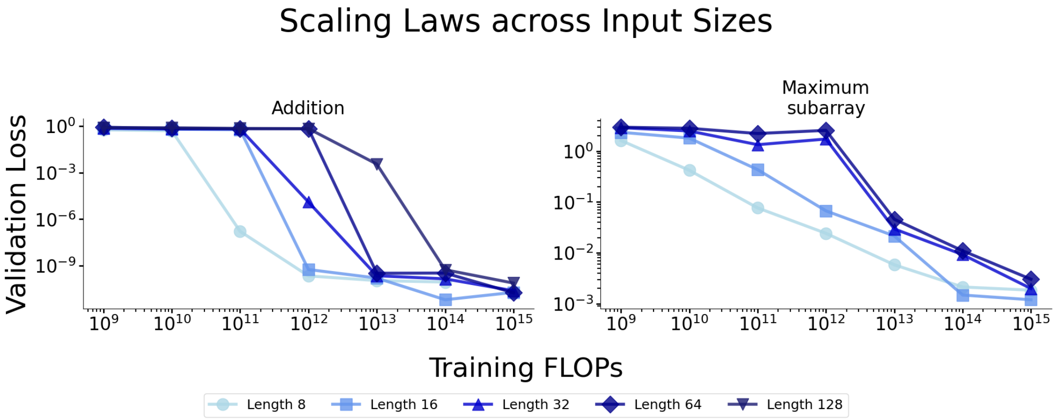

Figure 7: Models trained for addition exhibit phase transition for all task lengths. However, models trained for maximum subarray do not exhibit phase transition for smaller task lengths

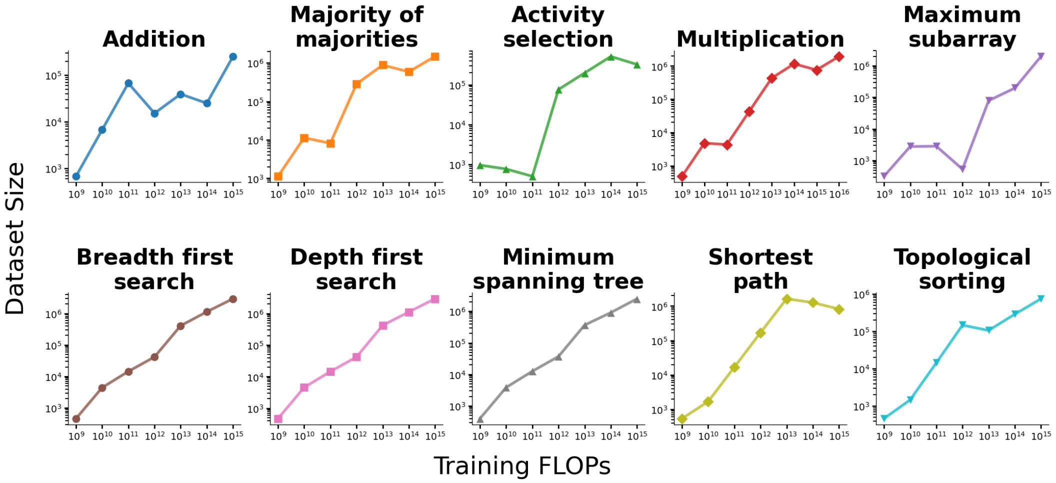

Figure 8: Compute-optimal dataset size (# of training examples) vs training FLOPs

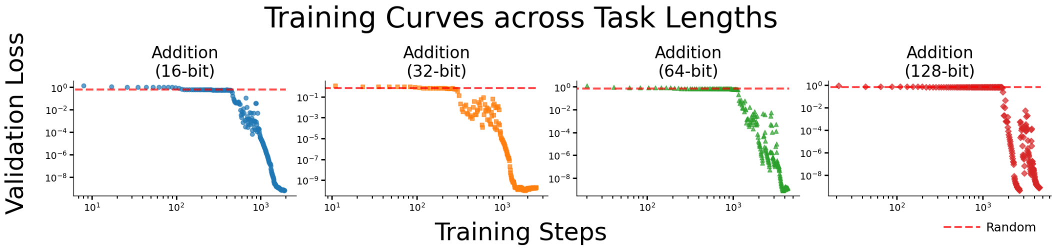

Figure 9: Models exhibit phase transitions for increasing task lengths. Compute-optimal training run is selected for the earliest training FLOPs in the same fashion as Figure 2