Discovering Hidden Algebraic Structures via Transformers with Rank-Aware Beam GRPO

Jaeha Lee [*1] **Gio Huh** [*2] **Ning Su** [3 1] **Tony Yue Tu** [4]

Abstract

We study the capabilities of small-scale transformer models in symbolic reasoning, focusing on the NP-hard algebraic task of multivariate polynomial decomposition, with widespread applications in science and engineering. Our approach includes a fine-grained synthetic data generation pipeline, supervised pretraining, beam search, evaluations for scaling behavior and generalizability, and a novel rank-aware reinforcement learning method called Beam Grouped Relative Policy Optimization (BGRPO), which improves accuracy while reducing inference compute by up to 75%. Additionally, our model demonstrates competitive performance in polynomial simplification, outperforming Mathematica in various cases.

1. Introduction

Transformers, initially developed for natural language processing (Vaswani et al., 2017), have shown remarkable versatility across diverse domains such as vision (Dosovitskiy et al., 2020) and protein folding (Jumper et al., 2021). More recently, their applications in formal reasoning, symbolic mathematics and algorithmic tasks start to gain traction. Several works have showcased transformer-based architectures' ability to tackle highly structured problems, including theorem proving (Polu & Sutskever, 2020; Trinh et al., 2024), integration (Lample & Charton, 2020), matrix multiplication (Fawzi et al., 2022) and equation solving (Drori et al., 2022).

In this work, we investigate the transformer's capacity for non-linear latent pattern discovery in the context of functional decomposition, i.e. decomposing a complex function as the composition of simpler sub-functions. In contrast to step-by-step logical deduction, or pattern recognition in data analysis, functional decomposition poses significant new challenges to the transformer, because the forms of the sub-functions that we try to discover can be totally hidden or obscured in the final compact form of the original function. Furthermore, it requires extreme precision without any margin of error. Unlike more forgiving classification tasks, the decomposition problem admits only a sparse set of correct solutions: even minor deviations in signs or coefficients can render outputs completely invalid.

Beyond its theoretical interest, functional decomposition has ubiquitous applications in software engineering (Tempero et al., 2024), systems biology (Mori et al., 2023), mechanical design (She et al., 2024), systems engineering (Hernandez et al., 2024) and digital logic design (Adamski et al., 2005; Lin et al., 2008), where capturing hidden substructures within high-dimensional functions leads to more tractable and efficient models. However, identifying a function's latent compositional structure requires models to look past surface-level correlations, attending instead to deep algebraic symmetries and invariants.

A particularly rich case of functional decomposition arises in multivariate polynomial functions. The polynomial decomposition problem over a ring k seeks to decompose a given polynomial $f \in k[x_1, \ldots, x_n]$ into polynomials $g \in k[y_1, \ldots, y_m]$

[*]Equal contribution [1]Walter Burke Institute for Theoretical Physics, California Institute of Technology, Pasadena, CA 91125, U.S.A. [2]Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, U.S.A. [3]Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA [4]Division of Physics, Mathematics and Astronomy, California Institute of Technology, Pasadena, CA 91125, U.S.A.. Correspondence to: Jaeha Lee <jaeha@caltech.edu>, Tony Yue Yu <yuyuetony@gmail.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

and $h_1, \ldots, h_m \in k[x_1, \ldots, x_n]$ such that

$$f(x_1, \ldots, x_n) = g\big(h_1(x_1, \ldots, x_n),\ \ldots\ ,\ h_m(x_1, \ldots, x_n)\big). \tag{1}$$

It has wide-ranging applications from cryptography (Patarin & Goubin, 1997) to dynamical modeling (Dang & Testylier, 2012), signal processing (Demirtas et al., 2012) and robotics (Elias & Wen, 2025; Manocha & Canny, 1992).

The multivariate polynomial decomposition problem has been proved to be NP-hard by Dickerson (Dickerson, 1987; 1993), although efficient algorithms for various special cases are discussed in (Gathen et al., 2003; Von Zur Gathen, 1990a;b; Faugère & Perret, 2009a;b; Zhao et al., 2012). To illustrate the difficulty of the problem for the models, let us consider the following expression

$$\begin{aligned} f =& 2a_1^3b_1^3 + 25a_1^2b_1^2 + 6a_1^2a_2b_2b_1^2 + 6a_1^2a_3b_3b_1^2 + 6a_1a_2^2b_2^2b_1 + 6a_1a_3^2b_3^2b_1 \\ &+ 96a_1b_1 + 50a_1a_2b_2b_1 + 50a_1a_3b_3b_1 + 12a_1a_2a_3b_2b_3b_1 + 2a_2^3b_2^3 + 2a_3^3b_3^3 \\ &+ 25a_2^2b_2^2 + 25a_3^2b_3^2 + 6a_2a_3^2b_2b_3^2 + 96a_2b_2 + 6a_2^2a_3b_2^2b_3 + 96a_3b_3 \\ &+ 50a_2a_3b_2b_3 + 12 \end{aligned}$$

It has a hidden $O(3)$-symmetry, which can be revealed by decomposing $f = g \circ h$, with $g(y) = y^2 + 2(4+y)^3$ and $h = a_1b_1 + a_2b_2 + a_3b_3$. This is a highly nontrivial task to identify the inner function h directly from the expanded form of f, as its structure becomes completely obscured after polynomial substitution, expansion and simplification. Even in this relatively constrained case where g is univariate, discovering the decomposition requires recognizing non-linear latent patterns across dozens of terms. When g becomes multivariate, the complexity increases substantially, making the problem even more challenging.

To tackle the polynomial decomposition problem, we develop a systematic approach with four key components. First, we create a backward synthetic data generation pipeline that allows fine-grained control over polynomial complexity involving range of coefficients, degree, and number of variables. Second, we train lightweight transformer models on these synthetic datasets using supervised learning and analyze how performance scales across four axes (performance complexity scaling, architecture scaling, distribution adaptation, search strategy analysis). Third, we discover that both multi-sampling and greedy search methods struggle with the sparse solution space of the polynomial decomposition problem, and we implement a beam search strategy to effectively extract the models' capabilities. Finally, to address the computational intensity of beam search, we develop a rank-aware variant of the Grouped Relative Policy Optimization (GRPO) reinforcement learning algorithm called BGRPO, which encodes rank information directly in the reward function. Fine-tuning with BGRPO improves accuracy while reducing beam search width by up to 50%, resulting in 75% lower computational requirements during inference. Additionally, our model demonstrates competitive performance in polynomial simplification, outperforming Mathematica in various cases. This underscores the potential of transformer models to complement and extend classical symbolic computation systems.

2. Method

2.1. Backward Synthetic Data Generation

We generate synthetic data for supervised learning using a backward approach, starting from the decomposed form. First, we generate the inner functions ($h_1, \ldots, h_m$ in Eq. (1)) and the outer function (g in Eq. (1)) with random monomial terms of bounded degree and random coefficients within a given range. Then, we obtain the composed function (f in Eq. (1)) via substitution, expansion, and term collection. See Appendix A for the detailed algorithm. For each generated instance, we create a training pair consisting of the expanded polynomial f as input and its decomposed components $\{g, h_1, \ldots, h_{v_{\text{outer}}}\}$ as the target output. The model is trained to minimize the standard negative log-likelihood loss function.

2.2. Beam Search

Beam search is a breadth-first search algorithm that approximates optimal decoding by keeping track of the k most probable sequences at each step (Freitag & Al-Onaizan, 2017). For each of the k current sequences, the algorithm considers the top-k token extensions per sequence. These k^2 candidate continuations are then ranked by the sum of log probabilities of all tokens in the sequence, and only the top-k sequences with the highest cumulative log probability are retained for the next step. In this paper, we refer to k as the beam width, and to the position (1st, 2nd, etc.) of an output in the final beam as its rank.

Our analysis across all model outputs identified a specific error pattern in polynomial decomposition: the model achieves approximately 90% accuracy for predicting non-sign tokens (operators, numbers, variables), but exhibits near-random performance for deciding between positive and negative signs. This creates a unique inference challenge where exploration needs to be constrained for high-confidence structural elements while simultaneously expanded for uncertain sign choices.

Beam search is particularly well-suited for this situation as it maintains the high-confidence structural backbone while systematically exploring variations in the uncertain components. Our experiments demonstrate that beam search significantly outperforms greedy decoding and random sampling for polynomial decomposition tasks. See Appendix G for a detailed error analysis and an explanation of beam search effectiveness for this task.

2.3. BGRPO : Reinforcement Learning Method Enhancing Beam Search Efficiency

The computational cost of beam search scales quadratically with beam width. There would be a significant computational advantage if we could improve the ranks of correct outputs. To address this, we introduce Beam Grouped Relative Policy Optimization (BGRPO), a reinforcement learning method that extends GRPO, uniquely taking into account rankings in the beam search, specifically designed for improving beam search inference efficiency.

Reinforcement learning enables models to explore solution spaces more effectively than supervised learning alone, enhancing the model's capabilities by addressing specific weaknesses through a reward mechanism. This approach encourages correct answers while discouraging incorrect ones based on an advantage function—the difference between a solution's reward and a baseline reward. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) estimates this baseline for each question by sampling a group of outputs, and has shown promising results for reinforcement learning in language generation tasks due to its sample efficiency and stability (DeepSeek-AI, 2025). We chose GRPO over traditional Proximal Policy Optimization (PPO) (Schulman et al., 2017) because it eliminates the need for a separate value network or reward model, reducing training complexity while improving stability, and its group-wise baseline calculation naturally fits tasks with a clear binary reward structure like polynomial decomposition.

Our proposed Beam Grouped Relative Policy Optimization (BGRPO) extends this approach by using beam search rather than independent sampling for generating the group of outputs. While this significantly alters the distribution of outputs, making their average reward less suitable as a traditional baseline, it still provides valid training signals by reinforcing correct answers and penalizing incorrect ones. BGRPO is particularly effective for our task because beam search generates outputs with identical structure that differ only in the confusing elements (signs), creating a focused learning signal.

Additionally, BGRPO incorporates rank information directly into the reward function by applying an exponential decay factor based on the position in the beam. This incentivizes correct answers to appear at earlier positions in the beam search, effectively pushing correct solutions toward the top of the beam ranking.

Training Objective For a prompt x, let $\mathcal{B}(x) = \{y_1, \dots, y_w\}$ be the set of beam search outputs with beam width w generated by the old policy $\pi_{\theta_{\text{old}}}$. Each output sequence y_i receives a reward r_i, where $r_i = 0$ for incorrect polynomial decomposition and $r_i = 1$ for correct decomposition. In BGRPO, we incorporate rank information by scaling the reward for correct decompositions using an exponential decay function $e^{-\text{rank}/w}$. We optimize the policy model π_θ for μ iterations by maximizing the following objective:

$$\mathcal{J}_{\text{BGRPO}}(\theta) = \frac{1}{w}\sum_{i=1}^{w}\left(\min\left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)}A_i, \text{clip}\left(\frac{\pi_\theta(y_i|x)}{\pi_{\theta_{\text{old}}}(y_i|x)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{\text{KL}}(\pi_\theta||\pi_{\text{ref}})\right), \tag{2}$$

where ε is the clipping parameter that constrains policy updates and β controls the KL divergence regularization term:

$$\mathbb{D}_{\text{KL}}(\pi_\theta||\pi_{\text{ref}}) = \frac{\pi_{\text{ref}}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{\text{ref}}(o_i|q)}{\pi_\theta(o_i|q)} - 1. \tag{3}$$

Here, π_{ref} is the reference policy, which is the initial model before BGRPO training. The advantage function A_i is computed without normalization as $A_i = r_i - \text{mean}(\{r_1, r_2, \cdots, r_w\})$, following the approach in (Liu et al., 2025).

2.4. Evaluation Axes

To systematically analyze our models' capabilities for the polynomial decomposition problem, we consider four key evaluation dimensions: Problem Complexity Scaling ($\mathcal{D}_1$), Architecture Scaling ($\mathcal{D}_2$), Distribution Adaptation ($\mathcal{D}_3$) and Search Strategy Analysis ($\mathcal{D}_4$). We refer to Appendix B for the details.

3. Selected Experimental Results

We refer to Appendix C for the details of our experimental setup, and to Appendix D for the complete results across the four evaluation axes mentioned above. Here we focus on the results of the first evaluation axis($\mathcal{D}_1$) and BGRPO. Additionally, we have a brief comparison with Mathematica.

3.1. Problem Complexity Scaling ($\mathcal{D}_1$)

In the first part of $\mathcal{D}_1$, we examine how model performance varies with the degrees of inner and outer polynomials. The result is shown in Figure 1. We use greedy search for the inference. Regardless of the degrees of the polynomials, our model achieves a remarkable single-output accuracy. Notably, when using beam search with a width of 10, the model's accuracy reaches 100% for these configurations.

Our analysis reveals a pattern: performance remains invariant to increases in the outer polynomial's degree, while decreasing when the inner polynomial's degree increases. This demonstrates that the transformer's decomposition capability is primarily limited by the complexity of the inner polynomial rather than that of the outer polynomial.

In the second part of $\mathcal{D}_1$, we investigate how the performance scales with v_{inner} and v_{outer}, the number of variables in the inner and outer polynomials. Figures 2 and 3 present these results.

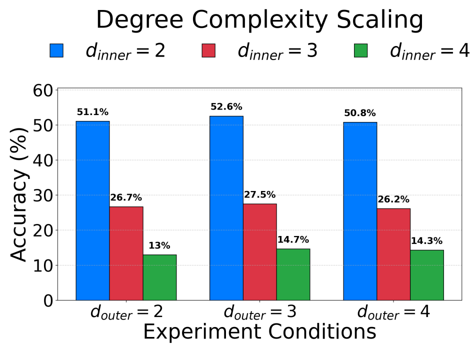

Figure 1. Performance across different $d_{\text{inner}}, d_{\text{outer}}$

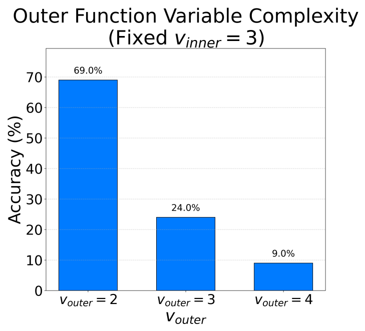

Figure 2. Performance across different v_{outer}

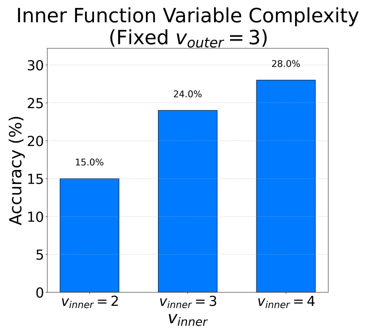

Figure 3. Performance across different v_{inner}

Given the challenging nature of multivariate polynomial decomposition, we evaluate the model's performance using beam search with a width of 30, considering a prediction correct if at least one of the 30 candidate outputs is correct decomposition.

Our results reveal two trends: performance decreases dramatically as v_{outer} increases, yet counter-intuitively improves as v_{inner} increases. This observation aligns with the following heuristic understanding: higher v_{outer} creates an information bottleneck, requiring the model to simultaneously resolve multiple interdependent inner functions. In contrast, higher v_{inner} provides more dimensions of input variation with additional structural indicators that can guide the decomposition process.

3.2. BGRPO Results

We evaluated BGRPO across models of varying sizes from our architecture scaling experiments($\mathcal{D}_2$), implementing versions both with and without rank signal. Fig 4 illustrates these results.

BGRPO consistently improved accuracy across all beam widths regardless of model size. Without rank signal, BGRPO gives average accuracy increases of 34.0%, 17.8%, and 12.4% for 6-layer models with dimension 256, 512, and 768 respectively. Including rank signal in BGRPO produces even more improvements, with average accuracy increases of 46.6%, 28.4%, and 30.2%.

These improvements translate to significant computational efficiency gains. For instance, the dimension-256 model initially achieved 26.1% accuracy with beam width 30. After applying BGRPO with rank signal, comparable accuracy (26.0%) was achieved with just beam width 16. This effectively halves the required beam width for equivalent performance. Since beam search computation scales quadratically with beam width, this improvement reduces beam search computation by approximately 75% while maintaining equivalent performance.

On average, BGRPO without rank signal reduced the required beam width by 31.3%, 14.9%, and 11.4% for 6-layer models with dimension 256, 512, and 768 respectively. When incorporating rank signal, BGRPO reduced required beam width even further, by 38.9%, 22.0%, and 26.5%.

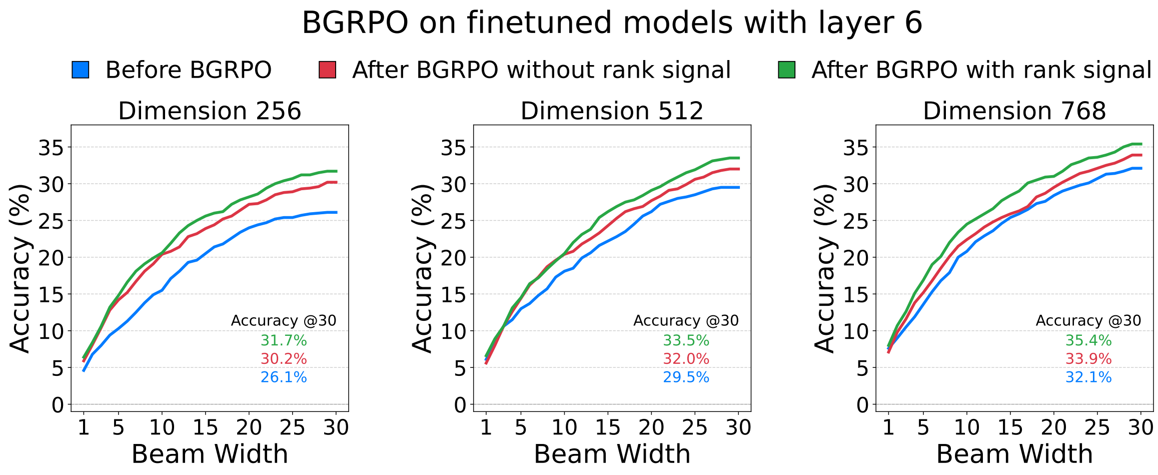

Figure 4. Accuracies on experiments with different dimension. Each experiment we have finetuned model with 2M data and models trained with BGRPO with and without rank signal on top of that.

3.3. Simplification Comparison with Mathematica

While polynomial simplification and polynomial decomposition represent two distinct mathematical objectives, simplification frequently arises as a consequence of decomposition, since decomposed forms generally exhibit reduced algebraic complexity compared to the original expression. In this subsection, we briefly explore the capabilities of our models for this related problem, and benchmark against the most powerful symbolic computation engine Mathematica. Despite our lightweight parameter budgets and the absence of any explicit simplification objective in our training, the models were able to reduce the leaf count (Wolfram Research, Inc., 1996) of complex expressions, with performance on par with — and in two of five complexity regimes surpassing —Mathematica's state-of-the-art FullSimplify function (see Appendix E).

Software and Data

Our code and data are publicly available at `https://github.com/Jaeha0526/PolynomialDecomposition`. The repository contains: (1) our synthetic data generation code for creating polynomial decomposition datasets with controlled complexity parameters; (2) the complete implementation of transformer-based polynomial decomposition models, including training pipelines built on minGPT; (3) evaluation scripts for all four experimental axes presented in the paper; (4) our Beam Grouped Relative Policy Optimization (BGRPO) algorithm implementation; and (5) scripts to reproduce all experiments from the paper. Additionally, we provide an example Jupyter notebook demonstrating our methods at `https://github.com/abhishekpanigrahi1996/MOSS/submission-72`. The codebase is designed to be modular and extensible, allowing researchers to easily experiment with different architectural configurations, data generation parameters, and search strategies.

Acknowledgments

We would like to express our gratitude to Aike Liu for initiating this project with NS and for her valuable contributions during the early stages. Her initial insights and discussions were instrumental in shaping this work. We thank David Simmons-Duffin for granting JL and NS access to computational resources that were essential for exploring and assessing the potential of this project. A significant portion of our experiments was conducted using the High-Performance Computing facilities at Caltech, supported by TYY. JL and NS's work at Caltech was supported in part by Simons Foundation grant 488657 (Simons Collaboration on the Nonperturbative Bootstrap).

References

Adamski, M. A., Karatkevich, A., Wegrzyn, M., Rawski, M., Łuba, T., Jachna, Z., and Tomaszewicz, P. The influence of functional decomposition on modern digital design process. *Design of Embedded Control Systems*, pp. 193–204, 2005.

Dang, T. and Testylier, R. Reachability analysis for polynomial dynamical systems using the bernstein expansion. *Reliab. Comput.*, 17(2):128–152, 2012.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Demirtas, S., Su, G., and Oppenheim, A. V. Sensitivity of polynomial composition and decomposition for signal processing applications. In *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pp. 391–395. IEEE, 2012.

Dickerson, M. T. Polynomial decomposition algorithms for multivariate polynomials. Technical report, Cornell University, 1987.

Dickerson, M. T. General polynomial decomposition and the s-1-decomposition are np-hard. *International Journal of Foundations of Computer Science*, 4(02):147–156, 1993.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Drori, I., Zhang, S., Shuttleworth, R., Tang, L., Lu, A., Ke, E., Liu, K., Chen, L., Tran, S., Cheng, N., et al. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119, 2022.

Elias, A. J. and Wen, J. T. Ik-geo: Unified robot inverse kinematics using subproblem decomposition. *Mechanism and Machine Theory*, 209:105971, 2025.

Faugère, J.-C. and Perret, L. An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography. *Journal of Symbolic Computation*, 44(12):1676–1689, 2009a.

Faugère, J.-C. and Perret, L. High order derivatives and decomposition of multivariate polynomials. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, pp. 207–214, 2009b.

Fawzi, A., Kozhasov, K., Goldblum, M., Behrmann, J., Zhang, C., Fuchs, F., Huang, P.-S., Li, L., and Kohli, P. Discovering faster matrix multiplication algorithms with reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2022.

Freitag, M. and Al-Onaizan, Y. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, 2017. doi: 10.18653/v1/w17-3207. URL http://dx.doi.org/10.18653/v1/W17-3207.

Gathen, J. v. z., Gutierrez, J., and Rubio, R. Multivariate polynomial decomposition. *Applicable Algebra in Engineering, Communication and Computing*, 14(1):11–31, 2003.

Hernandez, I., Watson, B. C., Weissburg, M. J., and Bras, B. Using functional decomposition to bridge the design gap between desired emergent multi-agent-system resilience and individual agent design. *Systems Engineering*, 27(5): 911–930, 2024.

Huang, L., Zhao, K., and Ma, M. When to finish? optimal beam search for neural text generation (modulo beam size), 2018. URL https://arxiv.org/abs/1809.00069.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Kaplan, J., McCandlish, S., Henighan, T., and Brown, T. B. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Karpathy, A. mingpt. https://github.com/karpathy/minGPT, 2020.

Lample, G. and Charton, F. Deep learning for symbolic mathematics. *arXiv preprint arXiv:2006.02974*, 2020.

Lin, H.-P., Jiang, J.-H. R., and Lee, R.-R. To sat or not to sat: Ashenhurst decomposition in a large scale. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 32–37. IEEE, 2008.

Liu, Z., Chen, C., Li, W., Qi, P., Pang, T., Du, C., Lee, W. S., and Lin, M. Understanding r1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

Manocha, D. and Canny, J. F. Real time inverse kinematics for general 6r manipulators. In *ICRA*, pp. 383–389, 1992.

Mori, M., Cheng, C., Taylor, B. R., Okano, H., and Hwa, T. Functional decomposition of metabolism allows a system-level quantification of fluxes and protein allocation towards specific metabolic functions. *Nature Communications*, 14(1):4161, 2023.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Patarin, J. and Goubin, L. Asymmetric cryptography with s-boxes is it easier than expected to design efficient asymmetric cryptosystems? In *International Conference on Information and Communications Security*, pp. 369–380. Springer, 1997.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. Sequence level training with recurrent neural networks, 2016. URL https://arxiv.org/abs/1511.06732.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

She, J., Belanger, E., and Bartels, C. Evaluating the effectiveness of functional decomposition in early-stage design: development and application of problem space exploration metrics. *Research in Engineering Design*, 35(3):311–327, 2024.

Tempero, E., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Kirk, D., Leinonen, J., Shakil, A., Sheehan, R., Tizard, J., Tu, Y.-C., et al. On the comprehensibility of functional decomposition: An empirical study. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, pp. 214–224, 2024.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

von Werra, L., Belkada, Y., Tunstall, L., Beeching, E., Thrush, T., Lambert, N., Huang, S., Rasul, K., and Gallouédec, Q. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Von Zur Gathen, J. Functional decomposition ofpolynomials: the tame case. *Journal of Symbolic Computation*, 9(3): 281–299, 1990a.

Von Zur Gathen, J. Functional decomposition of polynomials: the wild case. *Journal of Symbolic Computation*, 10(5): 437–452, 1990b.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022. URL https://arxiv.org/abs/2211.00593.

Wolfram Research, Inc. ComplexityFunction – Wolfram Language Documentation. https://reference.wolfram.com/language/ref/ComplexityFunction.html, 1996. Accessed 12 May 2025.

Zhao, S., Feng, R., and Gao, X.-S. On functional decomposition of multivariate polynomials with differentiation and homogenization. *Journal of Systems Science and Complexity*, 25(2):329–347, 2012.

A. Backward Synthetic Data Generation Details

Our synthetic data generation process provides fine-grained control over problem complexity through eight parameters: C_{inner} (coefficient range for inner polynomials), d_{inner} (maximum degree of inner polynomials), v_{inner} (number of variables in inner polynomials), t_{inner} (maximum number of terms in inner polynomials), and similarly C_{outer}, d_{outer}, v_{outer}, and t_{outer} for the outer polynomial.

Algorithm 1 Backward Generation of Synthetic Training Data

```
Require: Coefficient range $C_{\text{inner}}, C_{\text{outer}}$; maximal degrees $d_{\text{inner}}, d_{\text{outer}}$; variable counts $v_{\text{inner}}, v_{\text{outer}}$; term limits $t_{\text{inner}}$, $t_{\text{outer}}$.
1: Generate outer polynomial $g$ with $v_{\text{outer}}$ variables, coefficients $\in C_{\text{outer}}$, degree $= d_{\text{outer}}$, and no more than $t_{\text{outer}}$ monomial terms.
2: Generate $v_{\text{outer}}$ inner polynomials $h_1, \dots, h_{v_{\text{outer}}}$, where each $h_i$ has $v_{\text{inner}}$ variables, coefficients $\in C_{\text{inner}}$, degree $= d_{\text{inner}}$, and no more than $t_{\text{inner}}$ monomial terms.
3: $f \leftarrow g(h_1, \dots, h_{v_{\text{outer}}})$, i.e. substitute $h_1, \dots, h_{v_{\text{outer}}}$ into $g$, expand and collect the monomial terms.
4: return $(f,\ g,\ h_1, \dots, h_{v_{\text{outer}}})$
```

B. Evaluation Axes Details

To systematically analyze our models' capabilities for the polynomial decomposition problem, we consider four key evaluation dimensions.

Problem Complexity Scaling ($\mathcal{D}_1$). We analyze how the model performance varies with respect to changes in the complexity parameters for synthetic data generation. We vary the number of variables $v_{\text{inner}}, v_{\text{outer}}$, and the maximum degrees $d_{\text{inner}}, d_{\text{outer}}$ for both the inner and outer polynomials.

Architecture Scaling ($\mathcal{D}_2$). We investigate how model performance scales with key architectural hyperparameters of the transformer. In particular, we measure $\mathcal{P}(M(d, l, a))$, the performance of models with embedding dimension d, number of layers l, and number of attention heads a. Our goal is to characterize how these hyperparameters influence model capabilities.

Distribution Adaptation ($\mathcal{D}_3$). A practical challenge in applying transformers to symbolic computation is their sensitivity to the numerical ranges present in the training data. For example, models trained on specific coefficient ranges tend to struggle with polynomials outside these ranges. On the other hand, we found that models can rapidly adapt to new coefficient distributions with minimal additional training, suggesting that they manage to learn generalizable pattern recognition rather than merely memorizing specific numerical relationships.

To quantify the model's ability to transfer its polynomial decomposition skills to numerically distinct but structurally identical problems, we prepare the model $M^n_{C_1 \to C_2}$. This model is initially trained on 1M polynomial decomposition examples with $C_{\text{outer}} = C_1$ and then fine-tuned with n examples with $C_{\text{outer}} = C_2$ where $C_1 \cap C_2 = \emptyset$. We measure the performance of model $M^n_{C_1 \to C_2}$ on a test set of polynomial decomposition problems with $C_{\text{outer}} = C_2$:

$$\mathcal{G}(n) = \mathcal{P}\big(M^n_{C_1 \to C_2},\ \text{test set with } C_{\text{outer}} = C_2\big) \tag{4}$$

Search Strategy Analysis ($\mathcal{D}_4$). We investigate how beam search enhances model performance on polynomial decomposition tasks, analyzing its effectiveness across different model architectures and levels of problem complexity.

C. Experimental Setup

C.1. Synthetic Dataset Setup

For the axis $\mathcal{D}_1$ of the problem complexity scaling, we first examine degree scaling by training a model on 2M polynomial decomposition examples with different inner and outer degrees as described in Table 1. We then evaluate this model on separate test datasets with the same configuration parameters, each corresponding to one of nine different $(d_{\text{inner}}, d_{\text{outer}})$ pairs to assess performance across varying problem complexities.

For the second part of the $\mathcal{D}_1$ axis, we train a model for each combination of v_{inner} and v_{outer} varying from 2 to 4 while fixing the other parameter at 3. For each combination, we use 1M examples to train the model.

For the axis $\mathcal{D}_2$ of architecture scaling, we train multiple models with varying architectural configurations, all using the same dataset of 2M examples with polynomial parameters as described in Table 1.

For the axis $\mathcal{D}_3$ of distribution adaptation, we train initial models on 1M examples with $C_{\text{outer}} = C_1 = [-5, 5]$ and then adapt them to examples with $C_{\text{outer}} = C_2 = [-10, -6] \cup [6, 10]$. Other parameters are the same across both datasets as described in Table 1.

For the second part of $\mathcal{D}_1$ (Variable Scaling) and $\mathcal{D}_2$, we set $t_{\text{inner}} = t_{\text{outer}} = 3$ to prevent expressions from becoming too long. We describe our tokenization in Appendix F.

Table 1. Synthetic Dataset Configuration Across Evaluation Axes

Evaluation Axis	**Inner Coeff.**	**Outer Coeff.**	**Inner Degrees**	**Outer Degrees**	**Inner Vars**	**Outer Vars**
$\mathcal{D}_1$ (Degree Scaling)	$[-20, 20]$	$[-20, 20]$	$\{2, 3, 4\}$	$\{2, 3, 4\}$	1	1
$\mathcal{D}_1$ (Variable Scaling)	$[-5, 5]$	$[-5, 5]$	3	3	$\{2, 3, 4\}$	$\{2, 3, 4\}$
$\mathcal{D}_2$ (Architecture)	$[-5, 5]$	$[-5, 5]$	3	3	3	3
$\mathcal{D}_3$ (Adaptation)	$[-20, 20]$	$C_1 = [-5, 5]$	$\{1, 2\}$	$\{1, 2, 3, 4\}$	1	1
	$[-20, 20]$	$C_2 = [-10, -6] \cup [6, 10]$	$\{1, 2\}$	$\{1, 2, 3, 4\}$	1	1

C.2. Architecture Configuration

We employ a decoder-only transformer architecture following standard design principles (Vaswani et al., 2017). Table 2 summarizes our task-specific configurations across all experimental axes. For lightweight and effective training, we developed our own model and training pipeline based on `minGPT` (Karpathy, 2020).

Table 2. Transformer Model Configuration Across Experiments

Experiment	**Context Window**	**Embedding Dim.**	**Layers**	**Heads**
$\mathcal{D}_1$ (Degree Scaling)	256	512	6	8
$\mathcal{D}_1$ (Variable Scaling)	850	512	6	8
$\mathcal{D}_2$ (Architecture)	850	{256, 512, 768}	{4, 6}	8
$\mathcal{D}_2$ (Attention Heads)	850	512	6	{4, 8, 16}
$\mathcal{D}_3$ (Distribution)	256	512	4	8

Common settings: GELU activation, learned positional embeddings, multi-head attention with causal masking, MLP hidden dimension = $4\times$ embedding dimension.

C.3. Supervised Learning Details

We train our models using the Adam optimizer with an initial learning rate of 6×10^{-4}, incorporating a 10% warmup period followed by cosine decay. Each configuration initially trains on 1M instances, with additional 1M training examples added incrementally until performance saturation. We use a batch size of 200 throughout training. We train models with enough epochs until it saturates with the given dataset.

C.4. BGRPO Implementation

For the BGRPO reinforcement learning phase, we generate candidate solutions using beam search with a width of 32 and temperature of 1.0. We implement our approach using the GRPO functionality from the `trl` library (von Werra et al., 2020). The training process consists of 5 policy update iterations after sampling outputs for 8 distinct polynomial decomposition problems. We set the PPO clipping parameter ε to 0.2 and the KL divergence coefficient β to 0.01. The learning rate during BGRPO training is 1×10^{-5}. We train models from $\mathcal{D}_2$ on a dataset of 200 non-repeating problems, saving checkpoints every 5 iterations and selecting the best model based on performance with beam width 7.

D. Experimental Results Across Four Evaluation Axes

Here we presents complete experiment results.

D.1. Architecture Scaling ($\mathcal{D}_2$)

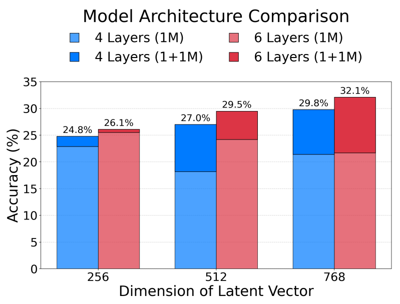

Figure 5. Accuracies on different number of layer and dimension.

In $\mathcal{D}_2$, we examine how model performance varies with architectural parameters: embedding dimension, number of layers, and number of attention heads. When varying the number of heads, we maintain a constant total embedding dimension, meaning that models with more heads have smaller per-head embedding dimensions. We use the dataset described in Section C.1 and evaluate using beam search with a width of 30.

Figure 5 reveals the scaling behavior (Kaplan et al., 2020) of transformer architectures on polynomial decomposition. As model capacity increases through higher embedding dimensions and additional layers, performance consistently improves.

Notably, our results demonstrate the presence of a data-dependent scaling threshold. With limited training data (1M examples), larger models initially underperform their simpler counterparts, particularly evident in the 6-layer configurations with higher embedding dimensions. However, this pattern reverses completely with additional training data, confirming that larger models possess superior capacity for mathematical pattern recognition when provided with sufficient examples to leverage their parametric advantage.

In $\mathcal{D}_2$, we also examine model performance with different numbers of attention heads. Our experiments reveal that increasing the number of attention heads while maintaining constant total embedding dimension leads to progressively deteriorating performance on polynomial decomposition tasks. Models with 4 heads achieved 32.0% accuracy, while those with 8 and 16 heads reached only 28.0% and 25.0% accuracy, respectively. This suggests that for our specific task of mathematical pattern recognition, fewer, more expressive attention heads with larger per-head dimensions provide better performance than numerous specialized heads with smaller dimensions.

D.2. Distribution Adaptation ($\mathcal{D}_3$)

We evaluate $\mathcal{G}(n)$ as defined in Eq. 4, which measures how quickly models adapt to new coefficient distributions as a function of adaptation sample size n. For this experiment, we train a model with 4 layers and 512 embedding dimension on the dataset described in Section C.1. The initial training used 1M examples with outer polynomial coefficient range C_1, followed by fine-tuning on n examples with coefficient range C_2 for a single epoch. We report the variance in accuracy based on three independent trials.

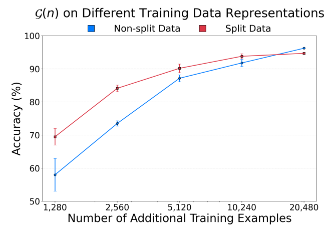

Figure 6. Performance recovery when adapting to a new coefficient distribution

Models trained exclusively on the first dataset achieve only 5.67% accuracy on the new distribution, despite reaching nearly 100% accuracy on the original distribution. Figure 6 illustrates how performance recovers during adaptation. Notably, despite using only $\approx 2\%$ of the original training data size, the model rapidly recovers its accuracy from single digits to over 90%. This rapid adaptation indicates successful transfer learning, suggesting that the model develops a general mathematical understanding of polynomial substructures rather than memorizing specific numerical relationships.

We further investigate whether alternative data representations could enhance this adaptation capability. We propose "split" representation of polynomials, where we randomly select terms from the expanded form and split their coefficients. For example:

$$\begin{aligned} f_{\text{non-split}}(a) &= -63 + 23a - 71a^2 - 11a^3 - 14a^4 - 12a^5 - 2a^6 \\ f_{\text{split}}(a) &= -63 + 23a - 4a^2 - 67a^2 - 8a^3 - 3a^3 - 7a^4 - 7a^4 - 12a^5 - a^6 - a^6 \end{aligned} \tag{5}$$

In Figure 6, the red line demonstrates $\mathcal{G}(n)$ of the model trained on data with both normal and split representation. Models trained on this mixed data including split representation demonstrate significantly faster adaptation, requiring only 70% of the additional training examples to reach equivalent performance on the new distribution.

This enhanced generalization likely stems from the model being forced to recognize mathematically equivalent but differently represented polynomials, compelling it to develop a deeper understanding of polynomial structure rather than memorizing specific patterns.

D.3. Search Strategy Analysis ($\mathcal{D}_4$)

We evaluate how search strategies impact model performance on polynomial decomposition tasks, with a particular focus on beam search efficiency. Figure 7 and 8 illustrate the accuracy achieved across different beam widths for polynomials with varying numbers of variables.

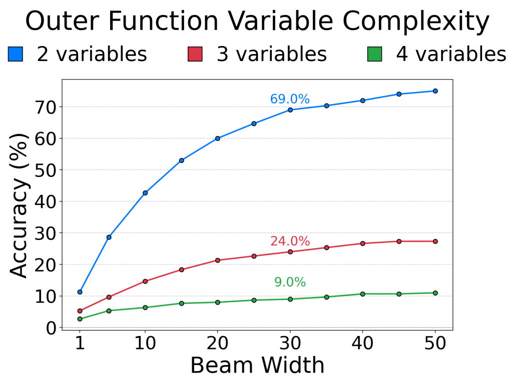

Figure 7. Beam width scaling with varying v_{outer} ($v_{\text{inner}} = 3$)

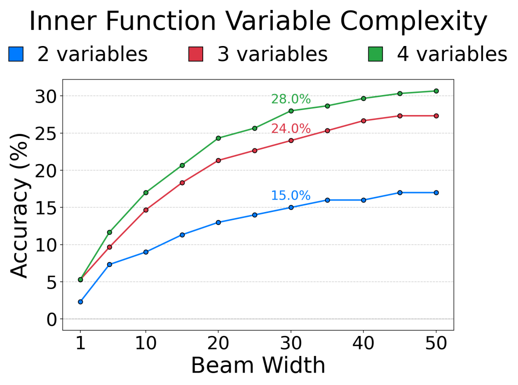

Figure 8. Beam width scaling with varying v_{inner} ($v_{\text{outer}} = 3$)

Our results reveal an unusually dramatic impact of beam search for polynomial decomposition compared to typical NLP tasks. For two-variable polynomials, accuracy improves from 11% with greedy search to 69% with a beam width of 30—a remarkable 6.3× improvement. This stands in stark contrast to standard neural machine translation applications, where beam search typically yields BLEU score improvements of only 2-4 points (Huang et al., 2018; Ranzato et al., 2016). Even more telling, most NMT systems show diminishing returns with beam widths beyond 5-10 (Freitag & Al-Onaizan, 2017).

E. Leaf Count Comparison with Mathematica

We compare the leaf counts of polynomial decompositions produced by our models with those generated by Mathematica's FullSimplify function.

Table 3. Average leaf count comparison (Beam width = 30)

Problem Complexity		**Leaf Count (mean)**		
v_O	v_S	Transformer	Mathematica	Δ
2	3	**27.28**	30.03	**-2.75**
3	3	22.85	**22.12**	**0.73**
4	3	22.52	20.00	2.52
3	2	17.27	**17.10**	**0.17**
3	4	**26.04**	27.56	**-1.52**

F. Tokenization

We encode polynomials using prefix notation, with separate tokens for operators, digits, and variables. Each number includes its sign, so we only use addition, multiplication, and power operators. Subtraction is represented as addition with a negative sign. Each input sequence consists of the tokenized expanded polynomial f followed by a question mark token '?'. The target output format depends on the number of outer variables: for $v_{\text{outer}} = 1$, the target output is simply the tokenized inner polynomial h; for $v_{\text{outer}} > 1$, the target output begins with the tokenized outer polynomial g followed by each tokenized inner polynomial $h_1, \ldots, h_{v_{\text{outer}}}$, with all polynomials separated by a delimiter token '&'.

Below is an example of a tokenized training input 'x' and target output 'y':

$$x : + * \mathrm{P}\ 9\ 0\ a + * \mathrm{N}\ 3\ 1\ 9\ \hat{}\ a\ \mathrm{P}\ 2 + * \mathrm{N}\ 3\ 6\ \hat{}\ a\ \mathrm{P}\ 3 * \mathrm{N}\ 1\ \hat{}\ a\ \mathrm{P}\ 4\ ? + \mathrm{N}\ 5 + * \mathrm{P}\, 1\, 8\, a\ \hat{}\ a\, \mathrm{P}\ 2 \,\square \ldots$$
$$y : \square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square + \mathrm{N}\ 5 + * \mathrm{P}\, 1\, 8\, a\ \hat{}\ a\, \mathrm{P} 2 \square\square \ldots$$

This example shows a training pair where the outer polynomial is $90a - 319a^2 - 36a^3 - a^4$ and the target inner polynomial is $-5 + 18a + a^2$. The $\square$ symbol represents a padding token which is excluded from the log-likelihood loss calculation.

G. Example Ouput Logits and Effectiveness of the Beam Search

Figure 9 shows example top-3 probabilities for each token position in the answer sequence at temperature 1, using the layer-6, embedding dimension 512 model from our $\mathcal{D}_2$ experiments. Correct answers are highlighted in red. The visualization clearly illustrates that the model's primary source of confusion occurs in sign decisions, while it confidently predicts most of the other token types.

Table 4 quantifies this observation by showing the probability and accuracy statistics for different token types across our model architectures from $\mathcal{D}_2$. These statistics were computed using a test set of 1000 polynomial decomposition problems at temperature 1.

Table 4. Token Type Analysis Across Different Model Architectures

Token Type	**Metric**	**4 Layers**			**6 Layers**		
		256 dim	**512 dim**	**768 dim**	**256 dim**	**512 dim**	**768 dim**
Sign	Probability	0.489 ± 0.001	0.489 ± 0.001	0.493 ± 0.001	0.491 ± 0.001	0.490 ± 0.001	0.490 ± 0.001
	Accuracy	0.519 ± 0.006	0.531 ± 0.006	0.530 ± 0.006	0.522 ± 0.006	0.523 ± 0.006	0.521 ± 0.006
Operator	Probability	0.920 ± 0.002	0.915 ± 0.002	0.919 ± 0.002	0.927 ± 0.002	0.925 ± 0.002	0.925 ± 0.002
	Accuracy	0.937 ± 0.002	0.934 ± 0.002	0.935 ± 0.002	0.943 ± 0.002	0.941 ± 0.002	0.942 ± 0.002
Number	Probability	0.880 ± 0.002	0.870 ± 0.002	0.878 ± 0.002	0.890 ± 0.002	0.885 ± 0.002	0.884 ± 0.002
	Accuracy	0.901 ± 0.002	0.893 ± 0.003	0.897 ± 0.002	0.911 ± 0.002	0.905 ± 0.002	0.903 ± 0.002

Table 4. Note: Values shown as mean ± standard error of the mean. The sign token probabilities are near-random, while operators and numbers show high confidence and accuracy.

As discussed in Section 2.2, our models achieve approximately 90% accuracy when predicting non-sign tokens, but exhibit near-random performance when choosing between positive and negative signs. This specific error pattern makes beam search particularly effective for our task.

The effectiveness of beam search stems from its ability to explore multiple sign configurations while preserving the high-confidence structural tokens. In probability terms, selecting a token with 0.1 probability instead of one with 0.9 probability is equivalent to making approximately 11 consecutive choices of a 0.45 probability token over a 0.55 probability token. Since our polynomial expressions typically contain fewer than 10 sign decisions, beam search with a width of approximately 30 can efficiently cover most viable sign permutations while maintaining the correct monomial structure identified with high confidence.

H. Attention Score Analysis: Monomial Heads

Attention mechanism analysis has provided valuable insights into transformer model behaviors, with studies identifying specialized attention heads that serve specific functions. For example, (Olsson et al., 2022) identified "Induction Heads" that play a crucial role in in-context learning, while (Wang et al., 2022) provided a comprehensive understanding of indirect object identification in GPT-2 Small.

In our analysis of attention patterns in polynomial decomposition models, we identified specialized attention heads that recognize the structure of polynomials, particularly focusing on monomial identification. We call these "Monomial Heads," and they appear consistently across all model sizes in our architecture scaling experiments ($\mathcal{D}_2$).

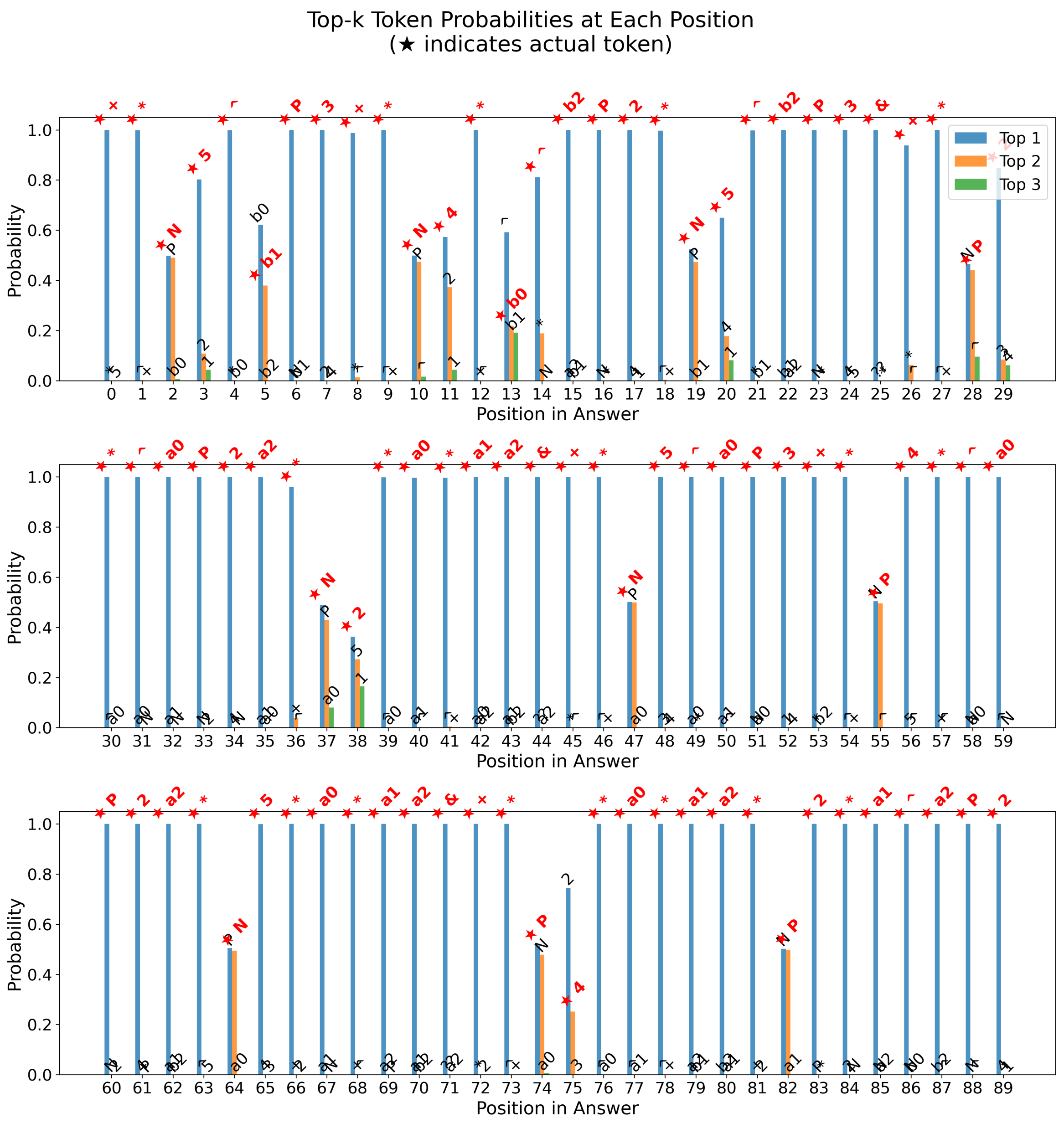

Figure 9. Top-3 probability for each token position in the answer sequence where

Answer: + * N 5 ^ b1 P 3 + * N 4 * b0 ^ b2 P 2 * N 5 ^ b2 P 3 & + * P 2 * ^ a0 P 2 a2 * N 2 * a0 * a1 a2 & + * N 5 ^ a0 P 3 + * P 4 * ^ a0 P 2 a2 * N 5 * a0 * a1 a2 & + * P 4 * a0 * a1 a2 * P 2 * a1 ^ a2 P 2

Question: + * P 6 2 5 ^ a0 P 9 + * N 1 5 0 0 * ^ a0 P 8 a2 + * P 1 8 7 5 * ^ a0 P 7 * a1 a2 + * P 1 2 0 0 * ^ a0 P 7 ^ a2 P 2 + * N 3 0 0 0 * ^ a0 P 6 * a1 ^ a2 P 2 + * P 1 8 7 5 * ^ a0 P 5 * ^ a1 P 2 ^ a2 P 2 + * N 3 2 0 * ^ a0 P 6 ^ a2 P 3 + * P 1 2 0 0 * ^ a0 P 5 * a1 ^ a2 P 3 + * N 1 6 2 8 * ^ a0 P 4 * ^ a1 P 2 ^ a2 P 3 + * P 4 3 3 * ^ a0 P 3 * ^ a1 P 3 ^ a2 P 3 + * N 1 2 8 * ^ a0 P 3 * ^ a1 P 2 ^ a2 P 4 + * N 3 5 2 * ^ a0 P 2 * ^ a1 P 3 ^ a2 P 4 + * N 3 2 * ^ a0 P 2 * ^ a1 P 2 ^ a2 P 5 + * N 2 0 8 * a0 * ^ a1 P 3 ^ a2 P 5 * N 4 0 * ^ a1 P 3 ^ a2 P 6 ?

Monomial Heads manifest in two distinct patterns in our models. First, in layer 0, several attention heads consistently attend to tokens 1-5 positions behind the current position, as shown in the leftmost plot of Figure 10. Second, in layer 1, we observe specialized behavior where certain heads focus attention on specific tokens within each monomial of the input polynomial (middle plot), while others specifically attend to delimiter tokens in the decomposition output (rightmost plot).

We hypothesize that this represents a two-stage process: in the first layer, the model identifies key tokens that serve as indicators for each monomial by examining local context (1-5 tokens behind). In the second layer, tokens within each monomial attend to these indicator tokens to establish their monomial membership. While this pattern is most clear in the encoding of the input polynomial, the decomposition output shows evidence of boundary recognition, particularly at the transitions between inner functions marked by delimiter tokens.

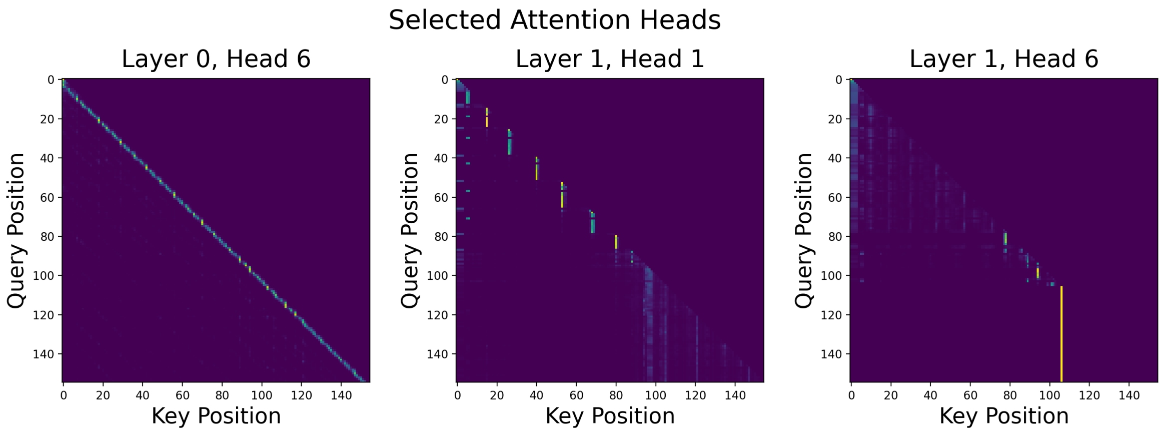

Figure 10. Attention score visualization of selected attention heads from our 6-layer transformer model with embedding dimension 768. The visualization shows attention patterns for a tokenized polynomial sequence and its decomposition.

Input polynomial: $+ * \text{P } 2\ 5\ 6 \wedge \text{a0 P } 9 + * \text{N } 1\ 9\ 2 * \wedge \text{a0 P } 8 \text{ a1} + * \text{P } 4\ 8 * \wedge \text{a0 P } 7 \wedge \text{a1 P } 2 + * \text{N } 4 * \wedge \text{a0 P } 6 \wedge \text{a1 P } 3 + * \text{N } 6\ 4 * \wedge \text{a0 P } 3 \wedge \text{a1 P } 6 + * \text{P } 1\ 6 * \wedge \text{a0 P } 2 \wedge \text{a1 P } 7 * \text{P } 6\ 4 \wedge \text{a1 P } 9\ ?$

Model's decomposition output: $+ * \text{N } 4 \wedge \text{b0 P } 3 + * \text{b0} \wedge \text{b2 P } 2 * \text{N } 1 \wedge \text{b2 P } 3\ \& + * \text{N } 4 \wedge \text{a0 P } 3 * \wedge \text{a0 P } 2 \text{ a1}\ \& + * \text{N } 3 \wedge \text{a1 P } 3 + * \text{N } 2 * \text{a1} \wedge \text{a2 P } 2 * \text{N } 4 \wedge \text{a2 P } 3\ \& * \text{N } 4 \wedge \text{a1 P } 3$ The visualization reveals how different attention heads focus on specific structural elements when decomposing polynomials.