Looped Transformers are Better at Learning Learning Algorithms

Liu Yang [1] **Kangwook Lee** [1] **Robert Nowak** [1] **Dimitris Papailiopoulos** [1]

Abstract

Transformers can "learn" to solve data-fitting problems generated by a variety of (latent) models, including linear models, sparse linear models, decision trees, and neural networks, as demonstrated by Garg et al. (2022). These tasks, which fall under well-defined function class learning problems, can be solved using iterative algorithms that involve repeatedly applying the same function to the input potentially an infinite number of times. In this work, we aim to train a transformer to emulate this iterative behavior by utilizing a *looped* transformer architecture (Giannou et al., 2023). Our experimental results reveal that the looped transformer performs equally well as the unlooped transformer in solving these numerical tasks, while also offering the advantage of having much fewer parameters.

1. Motivation

Tranformers (Vaswani et al., 2017; Brown et al., 2020; Devlin et al., 2019) have become the model of choice in the field of natural language processing (NLP) and other domains requiring sequence-to-sequence modeling. Their ability to learn *in-context* as an emergent property (Wei et al., 2022a) of large language models (LLMs) has garnered significant attention from the community (Min et al., 2022; Olsson et al., 2022; Li et al., 2023), yet the reason why LLMs are capable of learning in context remains unclear. In an effort to unravel this intriguing behavior of large language models (LLMs), Garg et al. (2022) take a step forward by delving into the investigation of how transformers behave when trained to tackle specific function class learning problems. Perhaps not surprisingly, transformers showcase exceptional performance across all examined tasks, almost matching or sometimes surpassing the capabilities of conventional solvers. One of the tasks they scrutinized is linear regression, i.e. solving $\boldsymbol{w}$ in $\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2$. To solve this linear regression problem, the transformer could, for instance, learn an approximation to the ordinary least squares solution on a single forward pass. The computation of the matrix inverse, as required in the ordinary least squares solution $(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$, is more difficult for transformers to learn compared to matrix multiplication (Charton, 2021; von Oswald et al., 2022). This is attributed to the increased number of layers and heads required in the inverse operation (Giannou et al., 2023). Nevertheless, gradient descent offers as an alternative solution for linear regression, which requires only the matrix multiplication: $\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$, but is applied repeatedly. The iterative and recursive nature of gradient descent aligns with the recently proposed *looped* transformer architecture (Giannou et al., 2023). Motivated by this observation, we ask the following question:

> Is it possible to train a looped transformer to acquire the iterative learning algorithm and attain comparable performance to standard (unlooped) transformers while utilizing fewer parameters?

The answer is positive. In the following sections, we will investigate the looped format in section 3.1. Subsequently, in section 4, we proceed to analyze and compare the empirical performance of unlooped and looped transformers.

2. Problem Setting

Let $\mathcal{F}$ denote a class of functions defined on $\mathbb{R}^d$. Let $\mathcal{D}_{\mathcal{F}}$ denote a probability distribution over $\mathcal{F}$ and let $\mathcal{D}_{\mathcal{X}}$ denote a probability distribution on $\mathbb{R}^d$. We generate a random learning *prompt* P as follows. A function f is sampled from $\mathcal{D}_{\mathcal{F}}$ and inputs $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k\}$ as well as the test sample $\boldsymbol{x}_{test}$ are sampled from $\mathcal{D}_{\mathcal{X}}$. The output of $\boldsymbol{x}_i$ is computed by $f(\boldsymbol{x}_i)$. The prompt is then $P = (\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{test})$ and the goal of a learning system is to predict $f(\boldsymbol{x}_{test})$. Let M be a learning system and let $M(P)$ denote its prediction (note it is not given f explicitly). The performance of M is measured by the squared error $\ell(M(P), f(\boldsymbol{x}_{test})) = (M(P) - f(\boldsymbol{x}_{test}))^2$. In this work, we focus on transformer-based learning systems and compare them with other known learning systems depending on the tasks. Specifically, we look at the GPT-2 decoder model (Radford et al., 2019) with L layer. By default $L = 12$ for the unlooped transformer, following Garg et al. (2022)'s setting, and $L = 1$ for the looped transformer.

[1]University of Wisconsin, Madison, WI. Correspondence to: Liu Yang <liu.yang@wisc.edu>.

Work presented at the ES-FoMo Workshop at ICML 2023.

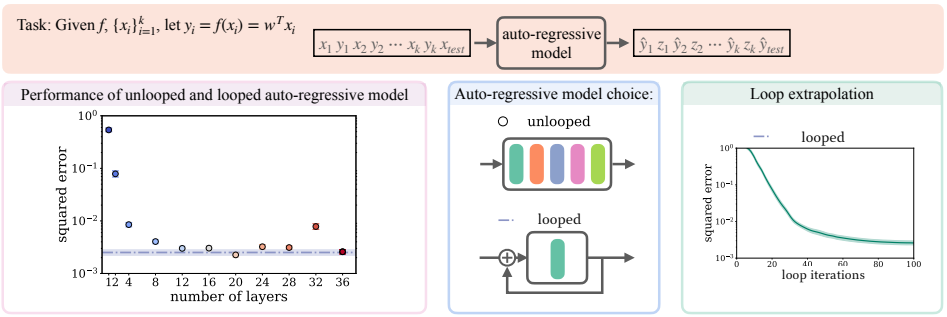


Figure 1. How to train a transformer to learn an iterative learning algorithm? Here we consider the task of training a transformer to solve linear regression tasks in context. The prompt $\{\boldsymbol{x}_1, y_1, \boldsymbol{x}_2, y_2, \cdots, \boldsymbol{x}_k, y_k, \boldsymbol{x}_{test}\}$ is passed into the auto-regressive model. As the auto-regressive model is designed to predict the next word, we anticipate that at $\boldsymbol{x}_i$, the model learns to predict the "next word," represented by y_i. However, since the $\boldsymbol{x}_i$'s are generated independently, the next word at y_i becomes unpredictable. Consequently, we disregard the output at y_i (denoted as $\boldsymbol{z}_i$ in the figure). The standard (unlooped) transformer utilized in this work is a decoder transformer model consisting of L layers, where each colored block represents a transformer layer. To learn an learning algorithm, we propose a *looped* transformer, where in each loop iteration, the input is added to the recurrence. On the Right, we can see *looped* transformer extrapolate the number of loop iterations, thus showcasing algorithmic behavior.

3. Looped Transformer Design

3.1. Looped Transformer with Inputs Injection

For an algorithm-emulated looped transformer, we anticipate the following characteristics in the looped architecture: 1) As loop iterations progress, the looped transformer should maintain or improve the performance; 2) the loop iterations have the potential to continue indefinitely without deterioration in performance. With slight reuse of the notation, let P be the inputs to the transformer model $M(\cdot)$, and let Y_t be the output after applying $M(\cdot)$ for t iterations. In a general form, a looped transformer can be represented as $Y_{t+1} = M(Y_t; P)$, $\forall t$. Lines of work (Lan et al., 2019; Dehghani et al., 2018) have investigated a specific variant known as the *weight-tying* form: $Y_{t+1} = M(Y_t)$ with $Y_0 = P$, and $t < T < \infty$ for some constant T. However, when considering the limit as t approaches infinity, the role of the initial input Y_0 diminishes, and the solution becomes essentially random or unpredictable (Bai et al., 2019; Bansal et al., 2022). To incorporate the input P into the solution of the looped transformer, we instead set $Y_{t+1} = M(Y_t + P)$. It's worth noting that the method of injecting the input is not restricted to addition alone. Experimental results have shown *looped* transformer match or outperform the unlooped transformer (Fig. 1 left), and extrapolate loop iterations (Fig. 1 right).

3.2. Training strategy

Following the problem setting in section 2, let the prompt to the transformer be $P = (\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_k, f(\boldsymbol{x}_k), \boldsymbol{x}_{test})$, with P^i denote the prompt prefix with i in-context samples $P^i = (\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_i, f(\boldsymbol{x}_i), \boldsymbol{x}_{i+1})$. The output of the *looped* transformer after t looping iterations is

$$Y_t(P^i|\theta) = \underbrace{M_\theta(M_\theta(\cdots M_\theta(}_{t \text{ iterations}} Y_0^i + P^i) + P^i) \cdots + P^i)$$

where the transformer M is parameterized by θ, and Y_0^i is a zero tensor with the same shape as P^i. Then we train the transformer by minimizing the following expected loss:

$$\min_\theta \mathbb{E}_P \left[\frac{1}{T} \sum_{t=b}^{T+b} \frac{1}{k+1} \sum_{i=0}^{k} \left(Y_t(P^i|\theta), f(\boldsymbol{x}_{i+1}) \right) \right],$$

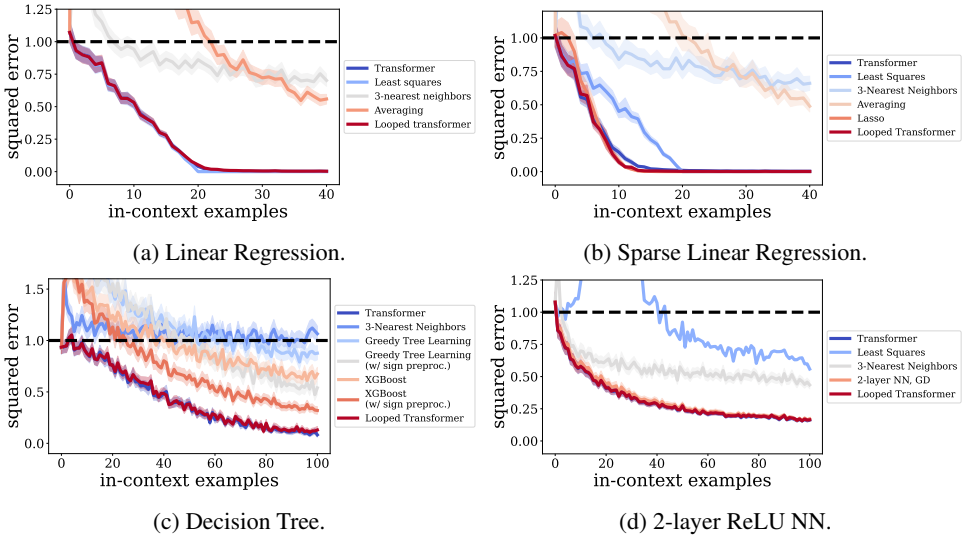
where we only measure the loss of the transformer over all prompt prefixes with loop iteration b to $b+T$. This truncated loss is inspired by the truncated backpropagation through time (TBPTT) (Hochreiter & Schmidhuber, 1997; Hinton & Sutskever, 2013) for computation saving.

4. Experimental Results

4.1. Performance on different tasks

We focus on the data-fitting problems generated by four latent models: 1) linear model with $d = 20$, $k = 100$; 2) sparse linear model with $d = 20$, $k = 100$, non-sparse entry $s = 3$; 3) decision tree with depth$= 4$, and input dimension $d = 20$; 4) 2 layer ReLU neural network with the hidden number of neurons to be 100, and input dimension $d = 20$. The parameters in the above functions are sampled from $\mathcal{N}(0, I_d)$, and the in-context samples $\boldsymbol{x}_i \sim \mathcal{N}(0, I_d)$ as well. The performance of the unlooped and *looped* transformer, along with other baselines, are shown in Fig. 2. Details of the baselines are in (Garg et al., 2022).

To compare with the unlooped transformer, we utilize an identical transformer as in (Garg et al., 2022) with the ex-



(a) Linear Regression.

(b) Sparse Linear Regression.

(c) Decision Tree.

(d) 2-layer ReLU NN.

Figure 2. Evaluating the trained transformer on in-context learning a) linear functions, b) sparse linear functions, c) random decision tree, and d) 2-layer ReLU Neural Network. The *looped* transformer demonstrates the ability to achieve performance on par with or even surpass the unlooped transformer (referred to as "Transformer" in the legend). This superior performance is accomplished while utilizing only 1/12th of the parameters employed by the unlooped transformer.

ception of the number of layers. Specifically, we employ a GPT-2 model with an embedding dimension of $D = 256$ and $h = 8$ attention heads, unlooped with $L = 12$ layers while looped with $L = 1$ layer. In terms of the number of parameters, the unlooped transformer uses 9.48M parameters[1], and the looped transformer uses 0.79M. We follow the same training strategy: train for $500,000$ iterations, with Adam optimizer, learning rate 0.0001. For the *looped* transformer, we set $T = 20$, and the curriculum for b will start with 0, and gradually increase to 350 for maximum. Both the unlooped and the *looped* transformer follow the training curriculum in terms of d and k, as specified in (Garg et al., 2022). When measuring the performance, we evaluate 1280 prompts and report the 90% confidence interval over 1000 bootstrap trails. Of all the tasks we looked at, the *looped* transformer constantly match the unlooped transformer, except for the sparse linear regression tasks, which we will discuss in the next paragraph.

Looped Transformer Does Better in Sparse Linear Regression Lasso (Tibshirani, 1996) is a strong baseline for solving sparse linear regression problems. In this experiment, we did a hyperparameter search over $\alpha \in \{1e-4, 1e-3, 0.01, 0.1, 1\}$, where α is the ℓ_1 parameter, and pick the best $\alpha = 0.01$, same as in (Garg et al., 2022). Notice that while the transformer achieves an error rate that nearly matches the Lasso performance, *looped* transformer achieves better than unlooped ones, and matches the Lasso performance almost perfectly. We conjuncture that this performance gain is due to the recurrent architecture of the *looped* transformer being better at learning the iterative behavior of the Lasso algorithm.

[1]The difference of the number of parameters reported here and in (Garg et al., 2022) (22.4M) are due to they also report the word-to-embedding parameters, despite not being utilized in the model. Hence, we exclude these parameters from the overall count.

4.2. Analyze Looped Transformer Model

In this section, we explore the impact of varying the number of layers (L) and embedding dimension (D) on *looped* transformer. These experiments specifically focus on the linear regression task, with $d = 5$ and $k = 10$, and training iterations 100,000. The remaining hyperparameters remain consistent with those in section 4.1.

In Fig. 3 (top), we plot the squared error for the unlooped transformer, and the looped transformer with L layers, applying t iterations. From the figure, we can see the larger L is, the faster it will converge. To achieve the performance of an unlooped transformer with L layers, the looped transformer needs more than L loop iterations. For instance, it took the looped transformer with 1 layer to loop for more than 50 iterations in order to achieve an 8-layer unlooped transformer's performance. This suggests the unlooped transformer and the *looped* transformer are learning different functions for each layer (iteration).

Figure 3 (bottem) illustrates the squared error of the *looped* transformer for various combinations of L and D. When $L = 1$, increasing the value of D results in improved convergence until reaching a saturation point at $D = 256$. Likewise, when $D = 256$ and L varies, the *looped* transformer

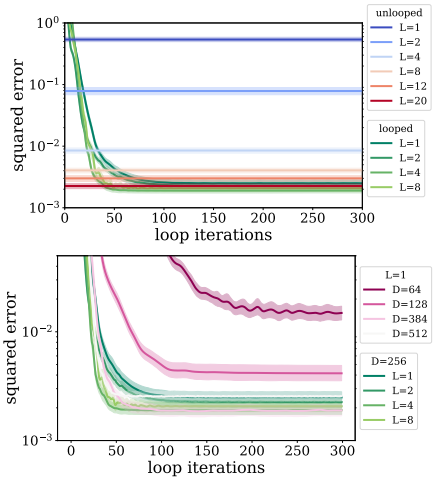


Figure 3. For linear function with $d = 5$, $k = 10$, we test on transformers with (top): unlooped and *looped* transformer with $D = 256$ and $h = 8$, but different L, and (bottom): *looped* transformer with different L and D, $h = 8$.

demonstrates comparable convergence performance, exhibiting differences primarily in the speed of convergence.

5. Discussion

Looped Transformer Achieve Algorithm-like Behavior by Finding a Fixed Point Solution. (Looped) transformer has the capability to approximate a solution, but not an exact solution (like Gradient Descent). In this study, we refrain from asserting that the looped transformer functions precisely like an algorithm, but instead conjecture it to find a fixed-point solution to the problem with a certain error rate. Its capacity to extrapolate the number of loops enables us to gain insights into how the looped transformer learns to solve recursively solvable tasks.

Training Efficiency. To measure the looped transformer at loop times b when b is large, we need to explicitly unroll the transformer for b iterations, which is very inefficient in terms of computation time. The selection of b and T depends on the complexity of the task and requires further investigation. In the future, our aim is to gain a deeper understanding of the appropriate choices for b and T, and develop a faster and accurate solver for the looped transformer.

Memory-Computation Trade-off During Inference. The computation cost associated with the looped transformer is determined by the number of loop iterations required to reach the minimum error (find the fixed-point solution). As shown in Fig. 3 (top), there is a memory-computation trade-off between the unlooped and the looped transformer. Also, Fig. 3 (bottom) illustrates the looped transformers with different layers L and embedding size D also exhibit different computation cost. In the future, we would like to also understand how L and D, and potentially the complexity of the task affect the computation cost of the looped transformer.

6. Related Works

Weight Sharing models Weight sharing is the nature in recurrent models such as RNN (Bengio et al., 2003) or LSTM (Hochreiter & Schmidhuber, 1997). These sequence modelings repeat the weights in the direction of sequence processing, enabling possibly infinite lengths of sequence inputs. Transformer (Vaswani et al., 2017; Devlin et al., 2019) typically has a fixed context length. In order to adjust to a longer context without parameter increase, several works (Dai et al., 2019; Wang et al., 2019; Hutchins et al., 2022) propose to combine recurrent models with transformers. Moreover, lines of works (Lan et al., 2019; Jaegle et al., 2021; Dehghani et al., 2018) also propose to share weights along the forward pass of the model. This weight sharing is across a limited number of layers, or with a halting criterion to exit the recurrence if needed. An extreme of the weight sharing along the function forward is the implicit model (Bai et al., 2019), where the function is repeatedly applied for an infinite amount of times. The applications of these implicit models are addressed in the next paragraph.

Deep Implicit Model Deep Implicit Models (Bai et al., 2018; 2019; 2020; Winston & Kolter, 2020) employ black-box solvers to find fixed-point solutions for implicit deep models. Later, Bai et al. (2021) proposed the Jacobian regularization to stabilize the training process. Nevertheless, this approach requires extensive hyperparameter tuning and still suffers from instability challenges. Implicit models are demonstrated to solve math problems with extrapolation ability (Decugis et al., 2022). This is a special case of using the recurrent model to solve math tasks, as we will examine more closely in the next paragraph.

Use Recurrent Models to Solve Math / Reasoning Task Several works (Zhang et al., 2022; Zhou et al., 2022b; Wei et al., 2022b; Zhou et al., 2022a; Bansal et al., 2022; Goel et al., 2022; Zhang et al., 2023) explore the ability of deep neural networks, including recurrent models, in solving mathematical or reasoning tasks that involve iterative processes. However, these math or reasoning tasks are with finite states/choices. Regression problem, on the other hand, is a relatively complex problem that involves continuous output space. Garg et al. (2022) investigate the transformer's ability in solving continuous function class problems, and later von Oswald et al. (2022) provide interpretations of this learning ability through linear attention heads. Our work, on the other hand, utilize the exact same architecture as in Garg et al. (2022): decoder model with non-linear attentions, and aim to provide empirical evidence and insights into the learning dynamics of this architecture.

References

Bai, S., Kolter, J. Z., and Koltun, V. Trellis networks for sequence modeling. *ArXiv*, abs/1810.06682, 2018.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *ArXiv*, abs/1909.01377, 2019.

Bai, S., Koltun, V., and Kolter, J. Z. Multiscale deep equilibrium models. *ArXiv*, abs/2006.08656, 2020.

Bai, S., Koltun, V., and Kolter, J. Z. Stabilizing equilibrium models by jacobian regularization. In *International Conference on Machine Learning*, 2021.

Bansal, A., Schwarzschild, A., Borgnia, E., Emam, Z. A. S., Huang, F., Goldblum, M., and Goldstein, T. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *ArXiv*, abs/2202.05826, 2022.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T. J., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

Charton, F. Linear algebra with transformers. *Trans. Mach. Learn. Res.*, 2022, 2021.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *ArXiv*, abs/1901.02860, 2019.

Decugis, J., Emerling, M., Ganesh, A., Tsai, A. Y., and Ghaoui, L. E. On the abilities of mathematical extrapolation with implicit models. 2022.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. Universal transformers. *ArXiv*, abs/1807.03819, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Garg, S., Tsipras, D., Liang, P., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *ArXiv*, abs/2208.01066, 2022.

Giannou, A., Rajput, S., yong Sohn, J., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. *ArXiv*, abs/2301.13196, 2023.

Goel, S., Kakade, S. M., Kalai, A. T., and Zhang, C. Recurrent convolutional neural networks learn succinct learning algorithms. *ArXiv*, abs/2209.00735, 2022.

Hinton, G. E. and Sutskever, I. Training recurrent neural networks. 2013.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

Hutchins, D. S., Schlag, I., Wu, Y., Dyer, E., and Neyshabur, B. Block-recurrent transformers. *ArXiv*, abs/2203.07852, 2022.

Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, 2021.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942, 2019.

Li, Y., Ildiz, M. E., Papailiopoulos, D., and Oymak, S. Transformers as algorithms: Generalization and stability in in-context learning. 2023.

Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. Rethinking the role of demonstrations: What makes in-context learning work? In *Conference on Empirical Methods in Natural Language Processing*, 2022.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T. J., 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. B., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *ArXiv*, abs/2209.11895, 2022.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the royal statistical society series b-methodological*, 58:267–288, 1996.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.

von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. *ArXiv*, abs/2212.07677, 2022.

Wang, Z., Ma, Y., Liu, Z., and Tang, J. R-transformer: Recurrent neural network enhanced transformer. *ArXiv*, abs/1907.05572, 2019.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., hsin Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models. *Trans. Mach. Learn. Res.*, 2022, 2022a.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., hsin Chi, E. H., Xia, F., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022b.

Winston, E. and Kolter, J. Z. Monotone operator equilibrium networks. *ArXiv*, abs/2006.08591, 2020.

Zhang, S. D., Tigges, C., Biderman, S. R., Raginsky, M., and Ringer, T. Can transformers learn to solve problems recursively? 2023.

Zhang, Y., Backurs, A., Bubeck, S., Eldan, R., Gunasekar, S., and Wagner, T. Unveiling transformers with lego: a synthetic reasoning task. *ArXiv*, abs/2206.04301, 2022.

Zhou, D., Scharli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., and hsin Chi, E. H. Least-to-most prompting enables complex reasoning in large language models. *ArXiv*, abs/2205.10625, 2022a.

Zhou, H., Nova, A., Larochelle, H., Courville, A. C., Neyshabur, B., and Sedghi, H. Teaching algorithmic reasoning via in-context learning. *ArXiv*, abs/2211.09066, 2022b.