On the Expressive Power of a Variant of the Looped Transformer

Anonymous authors
Paper under double-blind review

Abstract

Besides natural language processing, transformers exhibit extraordinary performance in solving broader applications, including scientific computing and computer vision. Previous works try to explain this from the expressive power and capability perspectives that standard transformers are capable of performing some algorithms. To empower transformers with algorithmic capabilities and motivated by the recently proposed looped transformer (Yang et al., 2024; Giannou et al., 2023), we design a novel transformer block, dubbed Algorithm Transformer (abbreviated as AlgoFormer). Compared with the standard transformer and vanilla looped transformer, the proposed AlgoFormer can perform efficiently in algorithm representation in some specific tasks. In particular, inspired by the structure of human-designed learning algorithms, our transformer block consists of a pre-transformer that is responsible for task preprocessing, a looped transformer for iterative optimization algorithms, and a post-transformer for producing the desired results after post-processing. We provide theoretical evidence of the expressive power of the AlgoFormer in solving some challenging problems, mirroring human-designed algorithms. Furthermore, some theoretical and empirical results are presented to show that the designed transformer has the potential to perform algorithm representation and learning. Experimental results demonstrate the empirical superiority of the proposed transformer in that it outperforms the standard transformer and vanilla looped transformer in some specific tasks. An extensive experiment on a real language task (neural machine translation of German and English) further validates the expressiveness and effectiveness of AlgoFormer.

1 Introduction

The emergence of the transformer architecture (Vaswani et al., 2017) marks the onset of a new era in natural language processing. Transformer-based large language models (LLMs), such as BERT (Devlin et al., 2019) and GPT-3 (Brown et al., 2020), revolutionized impactful language-centric applications, including language translation (Vaswani et al., 2017; Raffel et al., 2020), text completion/generation (Radford et al., 2019; Brown et al., 2020), sentiment analysis (Devlin et al., 2019), and mathematical reasoning (Imani et al., 2023; Yu et al., 2024). Beyond the initial surge in LLMs, these transformer-based models have found extensive applications in diverse domains such as computer vision (Dosovitskiy et al., 2021), time series (Li et al., 2019), bioinformatics (Zhang et al., 2023b), and addressing various physical problems (Cao, 2021). While many studies have concentrated on employing transformer-based models to tackle challenging real-world tasks, yielding superior performances compared to earlier models, the mathematical understanding of transformers remains incomplete.

The initial line of research interprets transformers as function approximators. Edelman et al. (2022) delve into the learnability and complexity of transformers for the class of sparse Boolean functions. Gurevych et al. (2022) explore the binary classification tasks, demonstrating that the transformer can circumvent the curse of dimensionality with a suitable hierarchical composition model for the posterior probability. Takakura & Suzuki (2023) investigate the approximation ability of transformers as sequence-to-sequence functions with infinite-dimensional inputs and outputs. They highlight that the transformer can avoid the curse of dimensionality under the smoothness conditions, due to the feature extraction and property sharing mechanism. However, viewing transformers solely as functional approximators, akin to feed-forward neural networks and convolutional neural networks, may not provide a fully satisfying understanding.

Another line of studies focuses on understanding transformers as algorithm learners. Garg et al. (2022) empirically investigate the performance of transformers in in-context learning, where the input tokens are input-label pairs generated from classical machine learning models, e.g., (sparse) linear regression and decision tree. They find that transformers can perform comparably as standard human-designed machine learning algorithms. Some subsequent works try to explain the phenomenon. Akyürek et al. (2023) characterize decoder-based transformer as employing stochastic gradient descent for linear regression. Bai et al. (2023) demonstrate that transformers can address statistical learning problems and employ algorithm selection, such as ridge regression, Lasso, and classification problems. Zhang et al. (2023a) and Huang et al. (2023) simplify the transformer model with reduced active parameters, yet reveal that the simplified transformer retains sufficient expressiveness for in-context linear regression problems. In Ahn et al. (2023), transformers are extended to implement preconditioned gradient descent. The looped transformer is proposed in Giannou et al. (2023), and is shown to have the potential to perform basic operations (e.g., addition and multiplication), as well as implicitly learn iterative algorithms Yang et al. (2024). More related and interesting studies can be found in Huang et al. (2023); Von Oswald et al. (2023); Mahankali et al. (2024).

In this paper, inspired by the recently proposed looped transformer (Yang et al., 2024; Giannou et al., 2023), we propose a novel transformer block, which we refer to as AlgoFormer, and strictly enforce it as an algorithm learner by regularizing its architecture. The transformer block consists of three sub-transformers, i.e., the pre-, looped, and post-transformers, designed to perform distinct roles. The pre-transformer is responsible for preprocessing the input data, and formulating it into some mathematical problems. The looped transformer acts as an iterative algorithm in solving the hidden problems. Finally, the post-transformer handles suitable postprocessing to produce the desired results. In contrast to standard transformers, the AlgoFormer is more likely to implement algorithms, due to its algorithmic structures shown in Figure 1.

Our main contributions can be summarized as follows:

- We introduce a novel transformer block (as shown in Figure 1), inspired by looped transformer (Yang et al., 2024; Giannou et al., 2023), namely the AlgoFormer. This block is designed as efficient algorithm learners, mimicking the structure of human-designed algorithms ; see Section 2.
- We theoretically show the expressive power of the AlgoFormer in solving three challenging tasks in implementing some practical human-designed algorithms (Theorems 3.1-3.3), including some task-specific pre- and post-processing, as well as the gradient descent; see Section 3.
- Beyond the gradient descent, we prove that the AlgoFormer can implement the (second-order) Newton's method in linear regression problems (Theorem 4.1). While our results primarily focus on encoder-based transformers, we also extend our findings to decoder-based transformers (Theorem 4.2); see Section 4.
- We experimentally investigate the behavior of the AlgoFormer with different hyperparameters in handling some challenging in-context learning tasks. The empirical results on both synthetic and real language tasks support the expressiveness and the powerfulness of the AlgoFormer; see Section 5.

This paper is organized as follows. The motivations for the design of the transformer block, including a brief introduction to transformer layer architectures and the algorithmic structure of the proposed transformer block, are presented in Section 2. We provide detailed results for the expressiveness of the designed AlgoFormer in tackling three challenging tasks, in Section 3. In Section 4, we additionally show that the designed transformer is capable of implementing Newton's method, a second-order optimization algorithm, beyond the gradient descent. We also extend our results for decoder-only transformers, which can only access data in the previous tokens for regression. In Section 5, experimental results studying the behavior of the designed transformer block are reported. We also empirically compare it with the standard transformer and vanilla looped transformer. Some concluding remarks and potential works are discussed in Section 6.

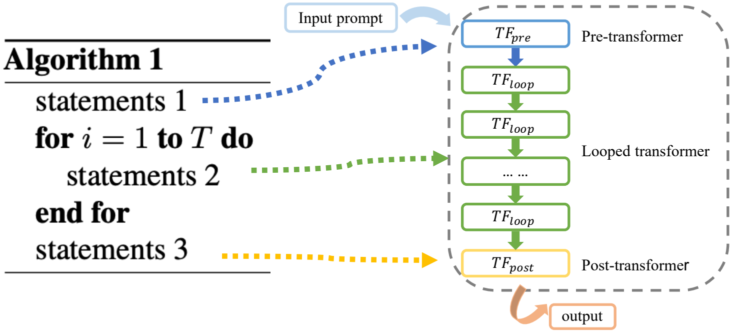


Figure 1: Algorithmic structure of the AlgoFormer. The pre-transformer conducts "statements 1" in the Algorithm; the looped transformer performs "statements 2" inner the for loop; the post-transformer carries out "statements 3". Here, $\mathrm{TF}_{\mathrm{pre}}$, $\mathrm{TF}_{\mathrm{loop}}$, and $\mathrm{TF}_{\mathrm{post}}$ are multi-layer transformers; "statements" represent some fundamental operations in classical algorithms.

2 Motivation

In this section, we mainly discuss the construction and intuition of the AlgoFormer. Its advantages over standard transformer is then conveyed. Before going into details, we first elaborate the mathematical definition of transformer layers.

2.1 Preliminaries

A One-layer transformer is mathematically formulated as:

$$\begin{aligned}
\mathrm{Attn}\left(\boldsymbol{X}\right) &= \boldsymbol{X} + \sum_{i=1}^{h} \boldsymbol{W}_V^{(i)} \boldsymbol{X} \cdot \mathrm{softmax}\left(\boldsymbol{X}^\top \boldsymbol{W}_K^{(i)\top} \boldsymbol{W}_Q^{(i)} \boldsymbol{X}\right), \\
\mathrm{TF}\left(\boldsymbol{X}\right) &= \mathrm{Attn}\left(\boldsymbol{X}\right) + \boldsymbol{W}_2 \mathrm{ReLU}\left(\boldsymbol{W}_1 \mathrm{Attn}\left(\boldsymbol{X}\right) + \boldsymbol{b}_1\right) + \boldsymbol{b}_2,
\end{aligned} \tag{1}$$

where $\boldsymbol{X} \in \mathbb{R}^{D \times N}$ is the input tokens; h is the number of heads; $\{\boldsymbol{W}_V^{(i)}, \boldsymbol{W}_K^{(i)}, \boldsymbol{W}_Q^{(i)}\}$ denote value, key and query matrices at i-th head, respectively; $\{\boldsymbol{W}_2, \boldsymbol{W}_1, \boldsymbol{b}_2, \boldsymbol{b}_1\}$ are parameters of the shallow feed-forward ReLU neural network. The attention layer with softmax activation function mostly exchanges information between different tokens by the attention mechanism. Subsequently, the feed-forward ReLU neural network applies nonlinear transformations to each token vector and extracts more complicated and versatile representations.

2.2 Algorithmic Structures of Transformers

As discussed in the introduction, rather than simply interpreting it as an implicit function approximator, the transformer may in-context execute some implicit algorithms learned from training data. However, it is still unverified that the standard multi-layer transformer is exactly performing algorithms.

Looped transformer. Equipped with the supposition that transformers can perform some basic operations, a shallow transformer is sufficient to represent iterative algorithms, as shown in the green part of Figure 1. The idea is first mentioned in Giannou et al. (2023), where they show that transformers can implement matrix addition and multiplication. Therefore, some iterative algorithms in scientific computing can be potentially realized by looped transformers. However, their construction does not explicitly imply the expressive power of transformers defined in Equation 1. Recently, Yang et al. (2024) empirically studied the behavior of looped transformers with different choices of hyperparameters in solving regression problems. Their experimental results further confirm the hypothesis that transformers are exactly learning iterative algorithms in solving linear regression problems, by strictly regularizing the transformer structure as looped transformers.

AlgoFormer. As shown in the green part of Figure 1, vanilla looped transformers (Yang et al., 2024; Giannou et al., 2023) admit the same structure as iterative algorithms. However, real applications are usually much more complicated. For example, given a task with data pairs, a well-trained researcher may first preprocess the data under some prior knowledge, and then formulate a mathematical (optimization) problem. Following that, some designed solvers, usually iterative algorithms, are performed. Finally, the desired results are obtained after further post-processing. The designed AlgoFormer (Algorithm Transformer), visualized in Figure 1, enjoys the same structure as wide classes of algorithms. Specifically, we separate the transformer block into three parts, i.e., pre-transformer TF_{pre}, looped transformer $\mathrm{TF}_{\text{loop}}$, and post-transformer $\mathrm{TF}_{\text{post}}$. Here, those three sub-transformers are standard multi-layer transformers in Equation (1). Given the input token vectors $\boldsymbol{X}$ and the number of iteration steps T, the output admits:

$$\mathrm{TF}_{\text{post}}\underbrace{\left(\mathrm{TF}_{\text{loop}}\left(\cdots\mathrm{TF}_{\text{loop}}\right.\right.}_{T\ \text{iterations}}\left.\left.(\mathrm{TF}_{\text{pre}}(\boldsymbol{X}))\right)\cdots\right). \tag{2}$$

Compared with standard transformers, the AlgoFormer acts more as the algorithm learner, by strictly regularizing the loop structure. In comparison to the results presented in the previous work Giannou et al. (2023), our construction of AlgoFormer is closer to the transformer in real applications. In their approach, the design of the looped transformer necessitates task-specific knowledge, involving operations like token order switching. In contrast, our model is task-independent, emphasizing the learning of algorithms solely from data rather than relying on potentially unknown prior knowledge. In contrast to the looped transformer in Yang et al. (2024), we introduce pre- and post-transformers, which play essential roles in real applications, and the designed transformer block can represent complex algorithms and solve challenging tasks more efficiently.

Expressiveness and efficiency. In algorithm representation (as shown in Figure 1), the vanilla looped transformer, the standard transformer, and the proposed AlgoFormer exhibit expressiveness regardless of efficiency (parameter sizes and computation). However, the AlgoFormer may enjoy more efficiency in some tasks. In specific algorithm representation with iterations, the standard transformer has redundant parameters, compared with the vanilla looped transformer and the AlgoFormer. For algorithms that require preprocessing and postprocessing, the AlgoFormer is more efficient for representation. We observe that the pre-transformer and the post-transformer in the AlgoFormer can be fused with the looped transformer. For example, suppose the pre-, looped, and post-transformers are all one-layer and one-head. In that case, the AlgoFormer can be realized by a one-layer and four-head looped transformer, where three heads contribute to perform pre-, looped, and post-transformer, and the additional head acts as the head selection. But the drawback is also obvious that the vanilla looped transformer with more heads introduces redundant computation.

3 Expressive Power

In this section, we theoretically show by construction that AlgoFormer is capable of solving some challenging tasks, akin to human-designed algorithms. The core idea is as follows. Initially, the pre-transformer undertakes the crucial task of preprocessing the input data, such as representation transformation. The looped transformer is responsible for iterative algorithms in optimization problems. Finally, it is ready to output the desired result by the post-transformer. Through the analysis of AlgoFormer's expressive power in addressing these tasks, we expect its potential to make contributions to the communities of scientific computing and machine learning.

3.1 Regression with Representation

We consider regression problems with representation, where the output behaves as a linear function of the input with a fixed representation function. Here, we adopt the L-layer MLPs with (leaky) ReLU activation function as the representation function $\Phi^*(\cdot)$. Specifically, we generate each in-context sample by first sampling the linear weight $\boldsymbol{A}$ from the prior $\mathcal{P}_A$, and then generating the input-label pair $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ with $\boldsymbol{x}_i \in \mathbb{R}^d \sim \mathcal{P}_x$, $\boldsymbol{y}_i = \boldsymbol{A}\Phi^*(\boldsymbol{x}_i) + \boldsymbol{\epsilon}_i$ and $\boldsymbol{\epsilon}_i \sim \mathcal{N}\left(\boldsymbol{0}, \sigma^2\boldsymbol{I}\right)$. We aim to find the test label $\boldsymbol{y}_{\text{test}} := \boldsymbol{A}\Phi^*(\boldsymbol{x}_{\text{test}})$, given the in-context samples and test data $\{\boldsymbol{x}_1, \boldsymbol{y}_1, \cdots, \boldsymbol{x}_N, \boldsymbol{y}_N, \boldsymbol{x}_{\text{test}}\}$. A reliable solver is expected first to identify the representation function and transform the input data $\boldsymbol{x}$ to its representation $\Phi^*(\boldsymbol{x})$. Then it reduces to a regression problem, and some optimization algorithms are performed to find the weight matrix from in-context samples. Finally, it outputs the desired result $\boldsymbol{y}_{\text{test}}$ by applying transformations on the test data. We prove by construction that there exists a AlgoFormer that solves the task, akin to the human-designed reliable solver.

Theorem 3.1. *There exists a designed AlgoFormer block with TF_{pre} (an $(L+1)$-layer two-head transformer), TF_{loop} (a one-layer two-head transformer), and TF_{post} (a one-layer one-head transformer), that outputs $\boldsymbol{A}\Phi^*(\boldsymbol{x}_{test})$ from the input-label pairs $\{\boldsymbol{x}_1, \boldsymbol{y}_1, \cdots, \boldsymbol{x}_N, \boldsymbol{y}_N, \boldsymbol{x}_{test}\}$ by fitting the representation function and applying gradient descent for multi-variate regression.*

Remarks. The detailed proof is available in Appendix A.1. Our construction of the transformer block involves three distinct sub-transformers, each assigned specific responsibilities. The pre-transformer, characterized by identity attention, is dedicated to representation transformation through feed-forward neural networks. This stage reduces the task to a multivariate regression problem. Subsequently, the looped transformer operates in-context to determine the optimal weight, effectively acting as an iterative solver. Finally, the post-transformer is responsible for the post-processing and generate the desired result $\boldsymbol{A}\Phi^*(\boldsymbol{x}_{\text{test}})$. Here, the input prompt to the transformer is formulated as

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{x}_{\text{test}} \\ \boldsymbol{0} & \boldsymbol{y}_1 & \cdots & \boldsymbol{0} & \boldsymbol{y}_N & \boldsymbol{0} \\ \boldsymbol{p}_1^x & \boldsymbol{p}_1^y & \cdots & \boldsymbol{p}_N^x & \boldsymbol{p}_N^y & \boldsymbol{p}_{N+1}^x \end{bmatrix},$$

where $\boldsymbol{p}_i^x$, and $\boldsymbol{p}_i^y$ denote positional embeddings and will be specified in the proof. Due to the differing dimensions of the input $\boldsymbol{x}$ and its corresponding label $\boldsymbol{y}$, zero padding is incorporated to reshape them into vectors of the same dimension. The structure of the prompt $\boldsymbol{P}$ aligns with similar formulations in previous works (Bai et al., 2023; Akyürek et al., 2023; Garg et al., 2022). For different input prompts $\boldsymbol{P}$, the hidden linear weights $\boldsymbol{A}$ are distinct but the representation function $\Phi^*(\cdot)$ is fixed. In comparison with the standard transformer adopted in Guo et al. (2024), which investigates similar tasks, the designed AlgoFormer has a significantly lower parameter size, making it closer to the envisioned human-designed algorithm. Notably, we construct the looped transformer to perform gradient descent for the multi-variate regression. However, the transformer exhibits remarkable versatility, as it has the capability to apply (ridge) regularized regression and more effective optimization algorithms beyond gradient descent. For more details, please refer to Section 4.

3.2 AR(q) with Representation

We consider the autoregressive model with representation. The dynamical (time series) system is generated by $\boldsymbol{x}_{t+1} = \boldsymbol{A}\Phi^*([\boldsymbol{x}_{t+1-q}, \cdots, \boldsymbol{x}_t]) + \boldsymbol{\epsilon}_t$, where $\Phi^*(\cdot)$ is a fixed representation function (e.g., we take the L-layer MLPs), and the weight $\boldsymbol{A}$ vary from different prompts. In standard AR(q) (multivariate autoregressive) models, the representation function $\Phi^*(\cdot)$ is identity. Here, we investigate a more challenging situation in which the representation function is fixed but unknown. A well-behaved solver should first find the representation function and then translate it into a modified autoregressive model. With standard Gaussian priors on the white noise $\boldsymbol{\epsilon}_t$, the Bayesian estimator of the AR(q) model parameters admits $\arg\max_{\boldsymbol{A}} \prod_{t=1}^N f(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}, \cdots, \boldsymbol{x}_{t-q}) = \arg\min_{\boldsymbol{A}} \sum_{t=1}^N \|\boldsymbol{x}_t - \boldsymbol{A}\Phi^*([\boldsymbol{x}_{t-q}, \cdots, \boldsymbol{x}_{t-1}])\|_2^2$, where $f(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}, \cdots, \boldsymbol{x}_{t-q})$ is the conditional density function of $\boldsymbol{x}_t$, given previous q observations. A practical solver initially identifies the representation function and transforms the input time series into its representation, denoted as $\Phi^*(\boldsymbol{x}_t)$. Then the problem is reduced to an autoregressive form. Similar to the previous subsection, we prove by construction that there exists a AlgoFormer, akin to human-designed algorithms, capable of effectively solving the given task.

Theorem 3.2. *There exists a designed AlgoFormer block with TF_{pre} (a one-layer q-head transformer with an $(L+1)$-layer one-head transformer), TF_{loop} (a one-layer two-head transformer), and TF_{post} (a one-layer one-head transformer), that predicts $\boldsymbol{x}_{N+1}$ from the data sequence $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$ by copying, transformation of the representation function and applying gradient descent for multi-variate regression.*

Remarks. The detailed proof can be found in Appendix A.2. The technical details are similar to Theorem 3.1. Additionally, the pre-transformer copies the feature from the previous q tokens, utilizing q heads for parallel processing.

3.3 Chain-of-Thought with MLPs

Chain-of-Thought (CoT) demonstrates exceptional performances in mathematical reasoning and text generation (Wei et al., 2022). The success of CoT has been theoretically explored, shedding light on its effectiveness in toy cases (Li et al., 2023) and on its computational complexity (Feng et al., 2023). In this subsection, we revisit the intriguing toy examples of CoT generated by leaky ReLU MLPs, denoted as CoT with MLPs, as discussed in Li et al. (2023). We begin by constructing an L-layer MLP with leaky ReLU activation. For an initial data point $\boldsymbol{x} \sim \mathcal{P}_x$, the CoT point $\boldsymbol{s}^\ell$ represents the output of the ℓ-th layer of the MLP. Consequently, the CoT sequence $\{\boldsymbol{x}, \boldsymbol{s}^1, \cdots, \boldsymbol{s}^L\}$ is exactly generated as the output of each (hidden) layer of the MLP. The target of CoT with MLPs problem is to find the next state $\hat{\boldsymbol{s}}^{\ell+1}$ based on the CoT samples $\{\boldsymbol{x}_1, \boldsymbol{s}_1^1, \cdots, \boldsymbol{s}_1^L, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N, \boldsymbol{s}_N^1, \cdots, \boldsymbol{s}_N^L, \boldsymbol{x}_{\text{test}}, \hat{\boldsymbol{s}}^1, \cdots, \hat{\boldsymbol{s}}^\ell\}$, where $\{\hat{\boldsymbol{s}}^1, \cdots, \hat{\boldsymbol{s}}^\ell\}$ denotes the CoT prompting of $\boldsymbol{x}_{\text{test}}$. We establish by construction in Theorem 3.3 that the AlgoFormer adeptly solves the CoT with MLPs problem, exhibiting a capability akin to human-designed algorithms.

Theorem 3.3. *There exists a designed AlgoFormer block with TF_{pre} (a seven-layer two-head transformer), TF_{loop} (a one-layer two-head transformer), and TF_{post} (a one-layer one-head transformer), that finds $\hat{\boldsymbol{s}}^{\ell+1}$ from samples $\{\boldsymbol{x}_1, \boldsymbol{s}_1^1, \cdots, \boldsymbol{s}_1^L, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N, \boldsymbol{s}_N^1, \cdots, \boldsymbol{s}_N^L, \boldsymbol{x}_{test}, \hat{\boldsymbol{s}}^1, \cdots, \hat{\boldsymbol{s}}^\ell\}$ by filtering and applying gradient descent for multi-variate regression.*

Remarks. We put the proof in Appendix A.3. The pre-transformer first identifies the positional number ℓ, and subsequently filters the input sequence into $\{\boldsymbol{s}_1^\ell, \boldsymbol{s}_1^{\ell+1}, \boldsymbol{s}_2^\ell, \boldsymbol{s}_2^{\ell+1}, \cdots, \boldsymbol{s}_N^\ell, \boldsymbol{s}_N^{\ell+1}, \hat{\boldsymbol{s}}^\ell\}$. This filtering transformation reduces the problem to a multi-variate regression problem. Compared with Li et al. (2023), where an assumption is made, we elaborate on the role of looped transformers in implementing gradient descent. While the CoT with MLPs may not be explicitly equivalent to CoT tasks in real applications, Theorem 3.3 somewhat implies the potential of the AlgoFormer in solving CoT-related problems.

4 Discussion

In this section, we provide complementary insights to the results discussed in Section 3. Firstly, as discussed in the remark following Theorem 3.1, we construct the looped transformer that employs gradient descent to solve (regularized) multi-variate regression problems. However, in practical scenarios, the adoption of more efficient optimization algorithms is often preferred. Investigating the expressive power of transformers beyond gradient descent is both intriguing and appealing. As stated in Theorem 4.1, we demonstrate that the AlgoFormer can proficiently implement Newton's method for solving linear regression problems. Secondly, the definition in Equation 1 implies the encoder-based transformer. In practical applications, a decoder-based transformer with causal attention, as seen in models like GPT-2 (Radford et al., 2019), may also be favored. For completeness, it is also compelling to examine the behavior of decoder-based transformers in algorithmic learning. Our findings, presented in Theorem 4.2, reveal that the decoder-based AlgoFormer can also implement gradient descent in linear regression problems. The primary distinction lies in the fact that the decoder-based transformer utilizes previously observed data to evaluate the gradient, while the encoder-based transformer calculates the gradient based on the full data samples.

4.1 Beyond the Gradient Descent

Newton's (second-order) methods enjoy superlinear convergence under some mild conditions, outperforming gradient descent with linear convergence. This raises a natural question:

Can the transformer implement algorithms beyond gradient descent, including higher-order optimization algorithms?

In this section, we address this question by demonstrating that the designed AlgoFormer can also realize Newton's method in regression problems.

Consider the linear regression problem given by:

$$\arg\min_{\boldsymbol{w}} \frac{1}{2N} \sum_{i=1}^{N} \left(\boldsymbol{w}^\top \boldsymbol{x}_i - y_i\right)^2. \tag{3}$$

Denote $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times d}$, $\boldsymbol{y} = [y_1, y_2, \cdots, y_N]^\top \in \mathbb{R}^{N \times 1}$ and $\boldsymbol{S} = \boldsymbol{X}^\top \boldsymbol{X}$. A typical Newton's method for linear regression problems follows the update scheme:

$$\boldsymbol{M}_0 = \alpha \boldsymbol{S}, \text{ where } \alpha \in \left(0, \frac{2}{\|\boldsymbol{S}\boldsymbol{S}^\top\|_2}\right], \ \boldsymbol{M}_{k+1} = 2\boldsymbol{M}_k - \boldsymbol{M}_k \boldsymbol{S} \boldsymbol{M}_k, \ \boldsymbol{w}_k^{\text{Newton}} = \boldsymbol{M}_k \boldsymbol{X}^\top \boldsymbol{y}. \tag{4}$$

As described in Söderström & Stewart (1974); Pan & Schreiber (1991), the above update scheme (Newton's method) enjoys superlinear convergence, in contrast to the linear convergence of gradient descent. The following theorem states that Newton's method in Equation 4 can be realized by the AlgoFormer.

Theorem 4.1. *There exists a designed AlgoFormer block with TF$_{pre}$ (a one-layer two-head transformer), TF$_{loop}$ (a one-layer two-head transformer), and TF$_{post}$ (a two-layer two-head transformer), that implements Newton's method described by Equation 4 in solving regression problems.*

Remarks. The proof can be found in Appendix A.4. The pre-transformer performs preparative tasks, such as copying from neighboring tokens. The looped-transformer is responsible for updating and calculating $\boldsymbol{M}_k \boldsymbol{x}_i$ for each token $\boldsymbol{x}_i$ at every step k. The post-transformer compute the final estimated weight $\boldsymbol{w}_T^{\text{Newton}}$ and outputs the desired the results $\boldsymbol{w}_T^{\text{Newton}\top} \boldsymbol{x}_{\text{test}}$, where T is the iteration number in Equation 2 and Equation 4. In a related study by Fu et al. (2023), similar topics are explored, indicating that transformers exactly perform higher-order optimization algorithms. However, our transformer architectures differ, and technical details are distinct.

4.2 Decoder-based Transformer

In the preceding analysis, the encoder-based AlgoFormer (with full attention) demonstrates its capability to solve problems by performing algorithms. Previous studies (Giannou et al., 2023; Bai et al., 2023; Zhang et al., 2023a; Huang et al., 2023; Ahn et al., 2023) also focus on the encoder-based models. We opted for an encoder-based transformer because full-batch data is available for estimating gradient and Hessian information. However, in practical applications, decoder-based models, like GPT-2, are sometimes more prevalent. In this subsection, we delve into the performance of the decoder-based model when executing iterative optimization algorithms, such as gradient descent, to solve regression problems.

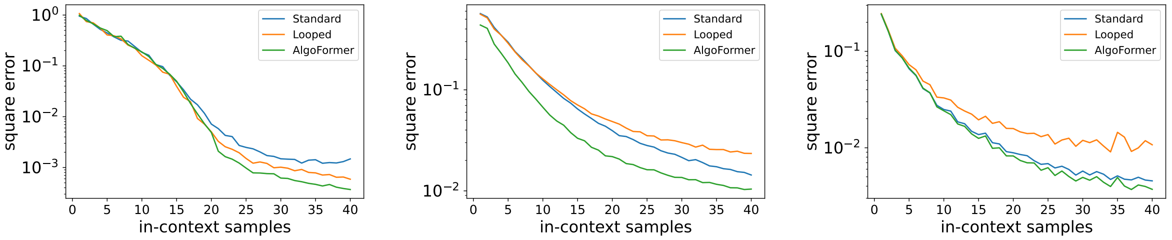
We consider the linear regression problem in Equation 3. Due to the limitations of the decoder-based transformer, which can only access previous tokens, implementing iterative algorithms based on the entire batch data is not feasible. However, it is important to note that the current token in a decoder-based transformer can access data from all previous tokens. To predict the label y_i based on the input prompt $\boldsymbol{P}^i = [\boldsymbol{x}_1, y_1, \cdots, \boldsymbol{x}_i]$, the empirical loss for the linear weight at $\boldsymbol{x}_i$ is given by

$$\boldsymbol{w}^i \in \arg\min_{\boldsymbol{w}} \mathcal{L}\left(\boldsymbol{w}; \boldsymbol{P}^i\right) := \frac{1}{2(i-1)} \sum_{j=1}^{i-1} \left(\boldsymbol{w}^\top \boldsymbol{x}_j - y_j\right)^2. \tag{5}$$

In essence, the linear weight is estimated using accessible data from the previous tokens, reflecting the restricted information available in the decoder-based transformer.

Theorem 4.2. *There exists a designed AlgoFormer block with TF$_{pre}$ (a one-layer two-head transformer), TF$_{loop}$ (a one-layer two-head transformer), and TF$_{post}$ (a two-layer two-head transformer), that outputs $\boldsymbol{w}_T^{i\top} \boldsymbol{x}_i$ for each input data $\boldsymbol{x}_i$, where $\boldsymbol{w}_T^i$ comes from $\arg\min_{\boldsymbol{w}} \mathcal{L}\left(\boldsymbol{w}; \boldsymbol{P}^i\right)$ after T steps of gradient descent.*

Remarks. The detailed proof is available in Appendix A.5. The technical details closely resemble those in Theorem 3.1, with the key distinction being that the decoder-based transformer can solely leverage data from previous tokens to determine the corresponding weight $\boldsymbol{w}^i$. Our findings align with those in Guo et al. (2024), although our transformer architectures and attention mechanisms differ. In a related study by Akyürek et al. (2023), similar topics are explored, demonstrating that the decoder-based transformer performs single-sample stochastic gradient descent, while our results exhibit greater strength with $\boldsymbol{w}^i \in \arg\min_{\boldsymbol{w}} \mathcal{L}\left(\boldsymbol{w}; \boldsymbol{P}^i\right)$. The construction of a decoder-based transformer for representing Newton's method is more challenging. We left it as a potential topic for future investigation.



(a) Regression with representation (b) AR(q) with representation (c) CoT with MLPs

Figure 2: The validation error of trained models (the standard transformer, the vanilla looped transformer, and the AlgoFormer), assessed on regression with representation, AR(q) with representation, and CoT with MLPs tasks. By choosing suitable hyperparameters (i.e., we set $(T, \Delta T) = (20, 15)$), the AlgoFormer has significantly better performance than the standard transformer and the vanilla looped transformer on those tasks.

5 Experiments

In this section, we conduct a comprehensive empirical evaluation of the performance of AlgoFormer in tackling challenging tasks, specifically addressing regression with representation, AR(q) with representation, and CoT with MLPs, as outlined in Section 2. Additionally, we also implement AlgoFormer on the neural machine translation task of German and English, demonstrating its expressiveness and effectiveness in the real-world language task.

5.1 Experimental Settings and Hyperparameters

Experimental settings. In all experiments, we adopt the decoder-based AlgoFormer, standard transformer (GPT-2), and vanilla looped transformer Yang et al. (2024). For synthetic tasks, we utilize $N = 40$ in-context samples as input prompts and $d = 20$ dimensional vectors with $D = 256$ dimensional positional embeddings for all experiments. To ensure fairness in comparisons, all models are trained using the Adam optimizer, with learning rate $\eta = 1e - 4$ and totally 500K iterations to ensure convergence. The standard transformer is designed to have $L = 12$ layers while pre-, looped and post-transformers are all implemented in one-layer. For task-specific settings and additional details, please refer to Appendix 5.1.

Training strategy. Our training strategy builds upon the methodology introduced in Yang et al. (2024). Let $\boldsymbol{P}^i = [\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_{i-1}, f(\boldsymbol{x}_{i-1}), \boldsymbol{x}_i]$ represents the input prompt for $1 \leq i \leq N$. We denote the AlgoFormer as $\mathrm{TF}^t_{\mathrm{Algo}}(\cdot; \boldsymbol{\Theta})$, where $f(\cdot)$ is a task-specific function and t indicates the number of loops (iterations) in Equation 2, and $\boldsymbol{\Theta}$ represent the transformer parameters. Instead of evaluating the loss solely on $\mathrm{TF}^T_{\mathrm{Algo}}(\cdot; \boldsymbol{\Theta})$ with T iterations, we minimize the expected loss over averaged iteration numbers:

$$\min_{\boldsymbol{\Theta}} \mathbb{E}_{\boldsymbol{P}} \left[\frac{1}{T - T_0} \sum_{t=T_0}^{T} \frac{1}{N} \sum_{i=1}^{N} \left\| \mathrm{TF}^t_{\mathrm{Algo}}(\boldsymbol{P}^i; \boldsymbol{\Theta}) - f(\boldsymbol{x}_i) \right\|_2^2 \right], \tag{6}$$

where $T_0 = \max\{T - \Delta T, 0\}$. To stabilize the training, we adopt the moving average over loop iterations from T_0 to T in the loss. The default setting for the AlgoFormer, as well as the vanilla looped transformer, involves setting $(T, \Delta T) = (20, 15)$. Here, the prompt formulation and the above loss may slightly differ for different tasks. For example, in the AR(q) task, the prompt is reformulated as $\boldsymbol{P}^i = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i]$ and $\boldsymbol{x}_{i+1} = f([\boldsymbol{x}_{i+1-q}, \cdots, \boldsymbol{x}_i])$ is the target for prediction. But the training strategy can be easily transmitted to other tasks. Here, both the iteration numbers T_0 and T are hyperparameters, which will be analyzed in the next subsection.

Regression with representation. In this task, we instantiate a 3-layer leaky ReLU MLPs, denoted as $\Phi^*(\cdot)$, which remains fixed across all tasks. The data generation process involves sampling a weight matrix $\boldsymbol{A} \in \mathbb{R}^{1\times 20}$. Subsequently, input-label pairs $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{40}$ are generated, where $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{20})$, $\epsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$

and $y_i = \boldsymbol{A}\Phi^*(\boldsymbol{x}_i) + \epsilon_i$. In Figure 2a, we specifically set $\sigma = 0$. Additionally, we explore the impact of different noise levels by considering $\sigma = 0.1$ and $\sigma = 1$.

AR(q) with representation. For this task, we set $q = 3$ and employ a 3-layer leaky ReLU MLP denoted as $\Phi^*(\cdot)$, consistent across all instances. The representation function accepts a 60-dimensional vector as input and produces 20-dimensional feature vectors. The time series sequence $\{\boldsymbol{x}_t\}_{t=1}^N$ is generated by initially sampling $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{20\times 20})$. Then the sequence is auto-regressively determined, with $\boldsymbol{x}_{t+1} = \boldsymbol{A}\Phi^*([\boldsymbol{x}_{t+1-q}, \cdots, \boldsymbol{x}_t]) + \boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{20})$.

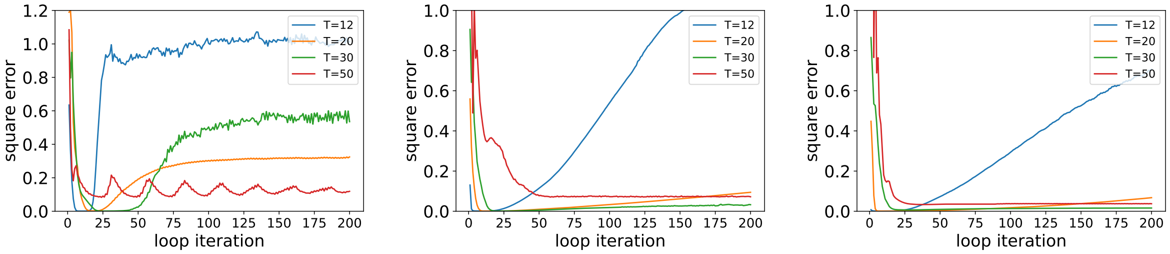
CoT with MLPs. In this example, we generate a 6-layer leaky ReLU MLP to serve as a CoT sequence generator, determining the length of CoT steps for each sample to be six. The CoT sequence, denoted as $\{\boldsymbol{x}, \boldsymbol{s}^1, \cdots, \boldsymbol{s}^L\}$ is generated by first sampling $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{20})$, where $\boldsymbol{s}^\ell \in \mathbb{R}^{20}$ represents the intermediate state output from the ℓ-th layer of the MLP.

5.2 AlgoFormer Exhibits High Expressiveness

In this subsection, we conduct a comparative analysis of the AlgoFormer against the standard and vanilla looped transformers across challenging tasks, as outlined in Section 3. Figure 2 illustrates the validation error trends, showcasing a decrease with an increasing number of in-context samples, aligning with our intuition. Crucially, the AlgoFormer consistently outperforms both the standard and the vanilla looped transformer across all tasks, highlighting its superior expressiveness in algorithm learning. Particularly in the CoT with MLPs task, both the AlgoFormer and the standard transformer significantly surpass the vanilla looped transformer. This further underscores the significance of preprocessing and postprocessing steps in handling complex real-world applications. The carefully designed algorithmic structure of the AlgoFormer emerges as an effective means of structural regularization, contributing to enhanced algorithm learning capabilities.

5.3 Impact of Hyperparameters

In this subsection, we conduct the empirical analysis of the impact of the hyperparameters on the AlgoFormer.

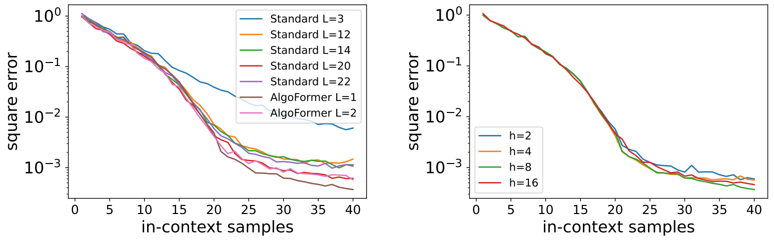


(a) $\Delta T = 5$ (b) $\Delta T = 10$ (c) $\Delta T = 15$

Figure 3: The validation error of trained models, evaluated on regression with representation task, with varying hyperparameters T and ΔT. The AlgoFormers are trained for T loops, defined in Equation 6, and the evaluation focuses on square loss at longer iterations, where the number of loop iterations far exceeds T.

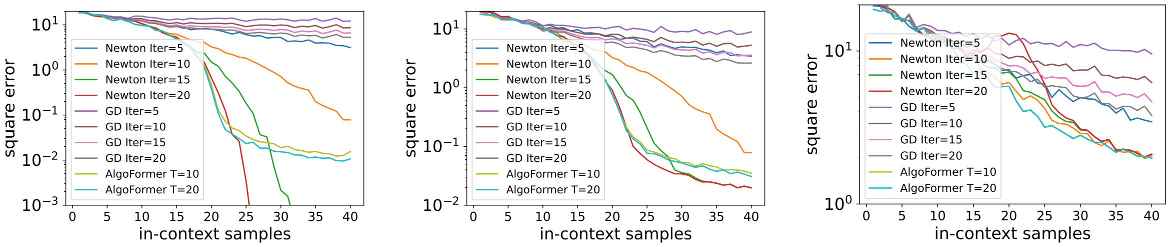
Loop iterations. We conduct comprehensive experiments on the AlgoFormer with varying loop numbers, on solving the regression with representation task. The results highlight the crucial role of both T and ΔT in the performance of the AlgoFormer. It is observed that a larger ΔT contributes to the stable inference of transformers. Comparing Figure 3a with Figures 3b and 3c, it is evident that a larger ΔT enhances stable long-term inference. The number of loop iterations T determines the model capacity in expressing algorithms. However, it is important to note that there exists a trade-off between the iteration numbers $(T, \Delta T)$ and computational costs. Larger $(T, \Delta T)$ certainly increases model capacity but also leads to higher computational costs and challenges in model training, as reflected in Figure 3.

Number of heads and layers. In our experiments on the AlgoFormer, we vary the numbers of heads and layers in TF_{loop} while addressing the regression with representation task. The results reveal a consistent



(a) Varying numbers of layers (b) Varying numbers of heads

Figure 4: The validation error of trained models, evaluated on regression with representation task, with varying numbers of layers (denoted as L) and heads (denoted as h). In the context of AlgoFormer, the number of layers L corresponds to the layers in the pre-, looped, and post-transformers, all of which are L-layer transformers. The AlgoFormers are trained with $(T, \Delta T) = (20, 15)$, defined in Equation 6.



(a) Linear regression, $\sigma = 0$ (b) Linear regression, $\sigma = 0.1$ (c) Linear regression, $\sigma = 1.0$

Figure 5: The validation error of trained AlgoFormer models and the linear regression models optimized by gradient descent and Newton's method. The AlgoFormers are trained with $(T, \Delta T) = (20, 15)$ and $(T, \Delta T) = (10, 10)$, defined in Equation 6.

trend that an increase in both the number of heads and layers leads to lower errors. This aligns with our intuitive understanding, as transformers with greater numbers of heads and layers exhibit enhanced expressiveness. However, a noteworthy observation is that 4-layer and 16-head transformers may exhibit suboptimal performance, possibly due to increased optimization challenges during model training. This finding underscores the importance of carefully selecting the model size, as a larger model, while offering higher expressiveness, may present additional training difficulties. The visualized results are shown in Figure 4. Moreover, compared with the standard transformer, even with the same number of layers, the AlgoFormer exhibits better performance in those tasks, mainly due to the introduced algorithmic structure. This finding highlights the role of the regularization of model structure. Therefore, we have reasons to believe that the good performance of the AlgoFormer not only comes from the higher expressiveness with deeper layers but also from the regularization of model architecture, which facilitates easier training and good generalization.

5.4 AlgoFormer and Human-Designed Algorithms

In this subsection, we compare the AlgoFormer with Newton's method and gradient descent in solving linear regression problems. We adopt the same default hyperparameters, with their selection grounded in a comprehensive grid search.

As illustrated in Figure 5, we observe that in the noiseless case, the AlgoFormer outperforms both Newton's method and gradient descent in the beginning stages. However, Newton's method suddenly achieves nearly zero loss (machine precision) later on, benefiting from its superlinear convergence. In contrast, our method maintains an error level around $1e-3$. With increasing noise levels, both Newton's method and gradient descent converge slowly, while our method exhibits better performance.

Several aspects contribute to this phenomenon. Firstly, in the noiseless case, Newton's method can precisely recover the weights through the linear regression objective in Equation 5, capitalizing on its superlinear convergence. On the other hand, the AlgoFormer operates as a black-box, trained from finite data. While we demonstrate good model expressiveness, the final generalization error of the trained transformer results from the model's expressiveness, the finite number of training samples, and the optimization error. Despite exhibiting high expressiveness, the trained AlgoFormer cannot eliminate the last two errors entirely. This observation resonates with similar findings in solving partial differential equations (Raissi et al., 2019) and large linear systems (Gu & Ng, 2023) using deep learning models.

Secondly, with larger noise levels, Newton's method shows suboptimal results. This is partly due to the inclusion of noise, which slows down the convergence rate, and Newton's method experiences convergence challenges when moving away from the local solution. In terms of global convergence, the AlgoFormer demonstrates superior performance compared to Newton's method.

Human-designed algorithms, backed by problem priors and precise computation, achieve irreplaceable performance. It's important to note that deep learning models, including transformers, are specifically designed for solving black-box tasks where there is limited prior knowledge but sufficient observation samples. We expect that transformers, with their substantial expressiveness, hold the potential to contribute to designing implicit algorithms in solving scientific computing tasks.

5.5 Applications to Language Tasks

In this subsection, we extend the evaluation of the proposed AlgoFormer to real-world language tasks, complementing its performance in real applications. Specifically, we focus on Neural Machine Translation using the IWSLT 2015 German-English dataset. The experimental setup includes a standard Transformer with 12 layers, 8 attention heads, a feature dimension of 256, and a learning rate of 5e-5. The pre-, looped, and post-transformers are all implemented as single layers, with T set to 12 and ΔT to 10 (see the hyperparameters in Equation 6 for training). Cross-entropy loss is used to evaluate the translation performance, where lower values indicate better results.

The results are presented in the table below:

Model	**Standard**	**Looped**	**AlgoFormer**
Cross entropy on validation set	4.994	4.726	**4.609**

As shown in the results, AlgoFormer outperforms both the standard Transformer and the vanilla looped Transformer in the Neural Machine Translation task, suggesting that conventional models may have inherent redundancies. This aligns with recent findings on the redundancy in large language models (Chen et al., 2024; Frantar & Alistarh, 2023; Xia et al., 2024). These preliminary results indicate the potential of AlgoFormer, which is designed as an algorithmic conductor, in real-world language tasks.

6 Conclusion and Future Works

In this paper, we introduce a novel AlgoFormer, an algorithm learner designed from the looped transformer, distinguished by its algorithmic structures. Comprising three sub-transformers, each playing a distinct role in algorithm learning, the AlgoFormer demonstrates expressiveness and efficiency while maintaining a low parameter size. Theoretical analysis establishes that the AlgoFormer can tackle challenging in-context learning tasks, mirroring human-designed algorithms. Our experiments further validate our claim, showing that the proposed transformer outperforms both the standard transformer and the vanilla looped transformer in specific algorithm learning and real-world language tasks.

There are some potential topics for future research. For example, AlgoFormer exhibits a strikingly similar structure to diffusion models. This similarity arises from the fact that both the AlgoFormer (algorithms) and diffusion models draw inspiration from dynamical systems. Consequently, an interesting question arises: can the empirical training techniques and theoretical insights from the well-explored diffusion model literature be translated to the AlgoFormer for understanding the transformer mechanism?

References

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=LziniAXEI9`.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=0g0X4H8yN4I`.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=liMSqUuVg9`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, 34:24924–24940, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/d0921d442ee91b896ad95059d13df618-Abstract.html`.

Xiaodong Chen, Yuxuan Hu, and Jing Zhang. Compressing large language models by streamlining the unimportant layer. *arXiv preprint arXiv:2403.19135*, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1:4171–4186, 2019. URL `https://aclanthology.org/N19-1423`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Benjamin L Edelman, Surbhi Goel, Sham Kakade, and Cyril Zhang. Inductive biases and variable creation in self-attention mechanisms. In *International Conference on Machine Learning*, pp. 5793–5831. PMLR, 2022. URL `https://proceedings.mlr.press/v162/edelman22a.html`.

Guhao Feng, Yuntian Gu, Bohang Zhang, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=qHrADgAdYu`.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Deqing Fu, Tian-Qi Chen, Robin Jia, and Vatsal Sharan. Transformers learn higher-order optimization methods for in-context learning: A study with linear models. *arXiv preprint arXiv:2310.17086*, 2023.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35: 30583–30598, 2022. URL `https://openreview.net/forum?id=flNZJ2eOet`.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, volume 202, pp. 11398–11442. PMLR, 2023. URL `https://proceedings.mlr.press/v202/giannou23a.html`.

Yiqi Gu and Michael K Ng. Deep neural networks for solving large linear systems arising from high-dimensional problems. *SIAM Journal on Scientific Computing*, 45(5):A2356–A2381, 2023.

Tianyu Guo, Wei Hu, Song Mei, Huan Wang, Caiming Xiong, Silvio Savarese, and Yu Bai. How do transformers learn in-context beyond simple functions? a case study on learning with representations. *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=ikwEDva1JZ`.

Iryna Gurevych, Michael Kohler, and Gözde Gül Şahin. On the rate of convergence of a classifier based on a transformer encoder. *IEEE Transactions on Information Theory*, 68(12):8139–8155, 2022.

Yu Huang, Yuan Cheng, and Yingbin Liang. In-context convergence of transformers. *arXiv preprint arXiv:2310.05249*, 2023.

Shima Imani, Liang Du, and Harsh Shrivastava. MathPrompter: Mathematical reasoning using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pp. 37–42. Association for Computational Linguistics, 2023. URL `https://aclanthology.org/2023.acl-industry.4`.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=Drrl2gcjzl`.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, 32, 2019. URL `https://papers.neurips.cc/paper_files/paper/2019/hash/6775a0635c302542da2c32aa19d86be0-Abstract.html`.

Yingcong Li, Kartik Sreenivasan, Angeliki Giannou, Dimitris Papailiopoulos, and Samet Oymak. Dissecting chain-of-thought: Compositionality through in-context filtering and learning. In *Advances in Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=xEhKwsqxMa`.

Arvind Mahankali, Tatsunori B Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=8p3fu56lKc`.

Santiago Ontanon, Joshua Ainslie, Zachary Fisher, and Vaclav Cvicek. Making transformers solve compositional tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3591–3607, 2022. URL `https://aclanthology.org/2022.acl-long.251`.

Victor Pan and Robert Schreiber. An improved newton iteration for the generalized inverse of a matrix, with applications. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1109–1130, 1991.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=R8sQPpGCv0`.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(1):5485–5551, 2020. URL `https://jmlr.org/papers/volume21/20-074/20-074.pdf`.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Torsten Söderström and GW Stewart. On the numerical properties of an iterative method for computing the moore–penrose generalized inverse. *SIAM Journal on Numerical Analysis*, 11(1):61–74, 1974.

Shokichi Takakura and Taiji Suzuki. Approximation and estimation ability of transformers for sequence-to-sequence functions with infinite dimensional input. In *International Conference on Machine Learning*, volume 202, pp. 33416–33447. PMLR, 2023. URL `https://proceedings.mlr.press/v202/takakura23a.html`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023. URL `https://proceedings.mlr.press/v202/von-oswald23a.html`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022. URL `https://openreview.net/forum?id=_VjQlMeSB_J`.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=09iOdaeOzp`.

Liu Yang, Kangwook Lee, Robert D Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=HHbRxoDTxE`.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=N8N0hgNDRt`.

Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *arXiv preprint arXiv:2306.09927*, 2023a.

Shuang Zhang, Rui Fan, Yuti Liu, Shuang Chen, Qiao Liu, and Wanwen Zeng. Applications of transformer-based language models in bioinformatics: a survey. *Bioinformatics Advances*, 3(1):vbad001, 2023b.

A Technical Proofs

In this section, we provide comprehensive proofs for all theorems stated in the main content.

Notation. We use boldface capital and lowercase letters to denote matrices and vectors respectively. Non-bold letters represent the elements of matrices or vectors, or scalars. For example, $A_{i,j}$ denotes the (i, j)-th element of the matrix $\boldsymbol{A}$. We use $\|\cdot\|_2$ to denote the 2-norm (or the maximal singular value) of a matrix.

A.1 Proof for Theorem 3.1

Positional embedding. The role of positional embedding is pivotal in the performance of transformers. Several studies have investigated its impact on natural language processing tasks, as referenced in (Kazemnejad et al., 2023; Ontanon et al., 2022; Press et al., 2022). In our theoretical construction, we deviate from empirical settings by using quasi-orthogonal vectors as positional embedding in each token vector. This choice, also employed by Li et al. (2023); Giannou et al. (2023), is made for theoretical convenience.

Lemma A.1 (Quasi-orthogonal vectors)**.** *For any fixed $\epsilon > 0$, there exists a set of vectors $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_N\}$ of dimension $\mathcal{O}(\log N)$ such that $\boldsymbol{p}_i^\top \boldsymbol{p}_i > \boldsymbol{p}_i^\top \boldsymbol{p}_j + \epsilon$ for all $i \neq j$.*

Before going through the details, the following lemma is crucial for understanding transformers as algorithm learners.

Lemma A.2. *A one-layer two-head transformer exhibits the capability to implement a single step of gradient descent in multivariate regression.*

Proof. Let us consider the input prompt with positional embedding as follows:

$$\boldsymbol{P} := \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{y}_1 & \cdots & \boldsymbol{0} & \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{0} \\ \frac{1}{N}\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \frac{1}{N}\boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}_k\boldsymbol{x}_1 - \boldsymbol{y}_1 & \boldsymbol{0} & \cdots & \boldsymbol{A}_k\boldsymbol{x}_N - \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{A}_k\boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 1 & 0 & \cdots & 1 & 0 & 1 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix},$$

where the 0-1 indicators are used to identify features, labels, and the test data, respectively. We denote the loss function for the multi-variate regression given samples $\{\boldsymbol{x}_1, \boldsymbol{y}_1, \cdots, \boldsymbol{x}_N, \boldsymbol{y}_N\}$ as

$$\mathcal{L}(\boldsymbol{A}) = \frac{1}{2N} \sum_{j=1}^{N} \|\boldsymbol{A}\boldsymbol{x}_j - \boldsymbol{y}_j\|_2^2,$$

then

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}(\boldsymbol{A}_k) = \frac{1}{N} \sum_{j=1}^{N} (\boldsymbol{A}_k\boldsymbol{x}_j - \boldsymbol{y}_j)\, \boldsymbol{x}_j^\top.$$

Now, let us define

$$\boldsymbol{W}_Q\boldsymbol{P} = \begin{bmatrix} c\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & c\boldsymbol{x}_N & \boldsymbol{0} & c\boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 1 & 1 & \cdots & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_K\boldsymbol{P} = \begin{bmatrix} \frac{1}{N}\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \frac{1}{N}\boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ 0 & 0 & \cdots & 0 & 0 & 0 & C \end{bmatrix},$$

and

$$\boldsymbol{W}_V\boldsymbol{P} = e^C \begin{bmatrix} \boldsymbol{A}_k\boldsymbol{x}_1 - \boldsymbol{y}_1 & \boldsymbol{0} & \cdots & \boldsymbol{A}_k\boldsymbol{x}_N - \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{A}_k\boldsymbol{x}_{\text{test}} & \boldsymbol{0} \end{bmatrix},$$

for some scalers $C, c > 0$. Here, we denote

$$\boldsymbol{Z} := \boldsymbol{P}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{P} = \begin{bmatrix} \frac{c}{N}\boldsymbol{x}_1^\top \boldsymbol{x}_1 & 0 & \cdots & \frac{c}{N}\boldsymbol{x}_1^\top \boldsymbol{x}_N & 0 & \frac{c}{N}\boldsymbol{x}_1^\top \boldsymbol{x}_{\text{test}} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \frac{c}{N}\boldsymbol{x}_N^\top \boldsymbol{x}_1 & 0 & \cdots & \frac{c}{N}\boldsymbol{x}_N^\top \boldsymbol{x}_N & 0 & \frac{c}{N}\boldsymbol{x}_N^\top \boldsymbol{x}_{\text{test}} & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ C & C & \cdots & C & C & C & C \end{bmatrix},$$

then

$$e^C \cdot \operatorname{softmax}\left(Z_{2i-1,2j-1}\right) \approx 1 + \frac{c}{N}\boldsymbol{x}_i^\top \boldsymbol{x}_j,$$

where the two sides of the "$\approx$" can be arbitrarily close if $C > 0$ is sufficiently large and $c > 0$ is sufficiently small. The constant here can be canceled by introducing another head. Therefore, the output of the attention layer is

$$\sum_{i=1}^{2} \boldsymbol{W}_V^{(i)} \boldsymbol{P} \cdot \operatorname{softmax}\left(\boldsymbol{P}^\top \boldsymbol{W}_K^{(i)\top} \boldsymbol{W}_Q^{(i)} \boldsymbol{P}\right) \approx c \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}(\boldsymbol{A}_k)\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}(\boldsymbol{A}_k)\boldsymbol{x}_N & \boldsymbol{0} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}(\boldsymbol{A}_k)\boldsymbol{x}_{\text{test}} & \boldsymbol{0} \end{bmatrix}.$$

The transformer layer's output, after passing through the feed-forward neural network, is expressed as:

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{y}_1 & \cdots & \boldsymbol{0} & \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}_{k+1}\boldsymbol{x}_1 - \boldsymbol{y}_1 & \boldsymbol{0} & \cdots & \boldsymbol{A}_{k+1}\boldsymbol{x}_N - \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{A}_{k+1}\boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 1 & 0 & \cdots & 1 & 0 & 1 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This signifies the completion of one step of gradient descent with $\boldsymbol{A}_{k+1} = \boldsymbol{A}_k - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{A}}(\boldsymbol{A}_k)$ and a positive step size $\eta > 0$. ☐

Proof for Theorem 3.1. We start by showing that L-layer transformer can represent L-layer MLPs. It is observed that the identity operation (i.e., $\operatorname{Attn}(\boldsymbol{X}) = \boldsymbol{X}$) can be achieved by setting $\boldsymbol{W}_V = \boldsymbol{0}$ due to the residual connection in the attention layer. Each feed-forward neural network in a transformer layer can represent a one-layer MLP. Consequently, the representation function $\Phi^*(\cdot)$ can be realized by L-layer transformers. At the output layer of the L-th layer transformer, let $\boldsymbol{A}_0$ be an initial guess for the weight. The current output token vectors are then given by:

$$\boldsymbol{P} := \begin{bmatrix} \Phi^*(\boldsymbol{x}_1) & \boldsymbol{0} & \cdots & \Phi^*(\boldsymbol{x}_N) & \boldsymbol{0} & \Phi^*(\boldsymbol{x}_{\text{test}}) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{y}_1 & \cdots & \boldsymbol{0} & \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A}_0\Phi^*(\boldsymbol{x}_1) & \boldsymbol{0} & \cdots & \boldsymbol{A}_0\Phi^*(\boldsymbol{x}_N) & \boldsymbol{0} & \boldsymbol{A}_{k+1}\Phi^*(\boldsymbol{x}_{\text{test}}) & \boldsymbol{0} \\ \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} & \frac{1}{N} & 0 & 0 \\ 1 & 0 & \cdots & 1 & 0 & 1 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Here, the set of quasi-orthogonal vectors $\{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_{N+1}\}$ is generated, according to Lemma A.1. The next transformer layer is designed to facilitate the exchange of information between neighboring tokens. Let

$$\boldsymbol{W}_K \boldsymbol{P} = \boldsymbol{W}_Q \boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1} \end{bmatrix}$$

and

$$\boldsymbol{W}_V \boldsymbol{P} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{y}_1 & \cdots & \boldsymbol{0} & \boldsymbol{y}_N & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix},$$

then

$$\boldsymbol{W}_V\boldsymbol{P}\cdot\operatorname{softmax}\left(\boldsymbol{P}^\top\boldsymbol{W}_K^\top\boldsymbol{W}_Q\boldsymbol{P}\right)\approx\left[\begin{array}{cccccccc}\frac{1}{2}\boldsymbol{y}_1 & \frac{1}{2}\boldsymbol{y}_1 & \cdots & \frac{1}{2}\boldsymbol{y}_N & \frac{1}{2}\boldsymbol{y}_N & \mathbf{0} & \mathbf{0}\end{array}\right],$$

where the two sides of the "≈" can be arbitrarily close if the temperature of the softmax function is sufficiently large, due to the nearly orthogonality of positional embedding vectors. It's important to note that the feed-forward neural network is capable of approximating nonlinear functions, such as multiplication. Here, we construct a shallow neural network that calculates the multiplication between the first d elements and the value $\frac{1}{N}$ in each token. Passing through the feed-forward neural network together with the indicators, we obtain the final output of the first $(L+1)$-layer transformer $\mathrm{TF}_{\mathrm{pre}}$:

$$\left[\begin{array}{ccccccc}\Phi^*\left(\boldsymbol{x}_1\right) & \mathbf{0} & \cdots & \Phi^*\left(\boldsymbol{x}_N\right) & \mathbf{0} & \Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right) & \mathbf{0}\\ \mathbf{0} & \boldsymbol{y}_1 & \cdots & \mathbf{0} & \boldsymbol{y}_N & \mathbf{0} & \mathbf{0}\\ \frac{1}{N}\Phi^*\left(\boldsymbol{x}_1\right) & \mathbf{0} & \cdots & \frac{1}{N}\Phi^*\left(\boldsymbol{x}_N\right) & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \boldsymbol{A}_0\Phi^*\left(\boldsymbol{x}_1\right)-\boldsymbol{y}_1 & \mathbf{0} & \cdots & \boldsymbol{A}_0\Phi^*\left(\boldsymbol{x}_N\right)-\boldsymbol{y}_N & \mathbf{0} & \boldsymbol{A}_0\Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right) & \mathbf{0}\\ \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1}\\ 1 & 0 & \cdots & 1 & 0 & 1 & 0\\ 0 & 1 & \cdots & 0 & 1 & 0 & 1\\ 0 & 0 & \cdots & 0 & 0 & 1 & 0\\ 0 & 0 & \cdots & 0 & 0 & 0 & 1\end{array}\right].$$

According to the construction outlined in Lemma A.2, there exists a one-layer, two-head transformer $\mathrm{TF}_{\mathrm{loop}}$, independent of the input data samples (tokens), that can implement gradient descent for finding the optimal weight $\boldsymbol{A}^*$ in the context of multivariate regression. The optimization aims to minimize the following empirical loss:

$$\min_{\boldsymbol{A}}\frac{1}{2N}\sum_{i=1}^{N}\left\|\boldsymbol{A}\Phi^*\left(\boldsymbol{x}_i\right)-\boldsymbol{y}_i\right\|_2^2.$$

After k-steps of looped transformer $\mathrm{TF}_{\mathrm{loop}}$, which corresponds to applying k steps of gradient descent, the resulting token vectors follows

$$\left[\begin{array}{ccccccc}\Phi^*\left(\boldsymbol{x}_1\right) & \mathbf{0} & \cdots & \Phi^*\left(\boldsymbol{x}_N\right) & \mathbf{0} & \Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right) & \mathbf{0}\\ \mathbf{0} & \boldsymbol{y}_1 & \cdots & \mathbf{0} & \boldsymbol{y}_N & \mathbf{0} & \mathbf{0}\\ \frac{1}{N}\Phi^*\left(\boldsymbol{x}_1\right) & \mathbf{0} & \cdots & \frac{1}{N}\Phi^*\left(\boldsymbol{x}_N\right) & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_1\right)-\boldsymbol{y}_1 & \mathbf{0} & \cdots & \boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_N\right)-\boldsymbol{y}_N & \mathbf{0} & \boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right) & \mathbf{0}\\ \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1}\\ 1 & 0 & \cdots & 1 & 0 & 1 & 0\\ 0 & 1 & \cdots & 0 & 1 & 0 & 1\\ 0 & 0 & \cdots & 0 & 0 & 1 & 0\\ 0 & 0 & \cdots & 0 & 0 & 0 & 1\end{array}\right].$$

These token vectors are then ready for processing by the output transformer layer $\mathrm{TF}_{\mathrm{post}}$. The post-transformer is designed to facilitate communication between the last two tokens and position the desired result $\boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right)$ in the appropriate position. We can similarly set

$$\boldsymbol{W}_K\boldsymbol{P}=\boldsymbol{W}_Q\boldsymbol{P}=\left[\begin{array}{ccccccc}\boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1}\end{array}\right],$$

$$\boldsymbol{W}_V\boldsymbol{P}=\left[\begin{array}{ccccccc}\boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_1\right)-\boldsymbol{y}_1 & \mathbf{0} & \cdots & \boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_N\right)-\boldsymbol{y}_N & \mathbf{0} & \boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right) & \mathbf{0}\end{array}\right],$$

and pass it through the feed-forward neural network. This results in the final output:

$$\left[\begin{array}{ccccccc}\Phi^*\left(\boldsymbol{x}_1\right) & \mathbf{0} & \cdots & \Phi^*\left(\boldsymbol{x}_N\right) & \mathbf{0} & \Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right) & \mathbf{0}\\ \mathbf{0} & \boldsymbol{y}_1 & \cdots & \mathbf{0} & \boldsymbol{y}_N & \mathbf{0} & \boldsymbol{A}_k\Phi^*\left(\boldsymbol{x}_{\mathrm{test}}\right)\end{array}\right],$$

which completes the proof.

A.2 Proof for Theorem 3.2

The following lemma highlights the intrinsic "copying" capability of transformers, a pivotal feature for autoregressive models, especially in the context of time series analysis.

Lemma A.3. *A one-layer transformer with q heads possesses the ability to effectively copy information from the previous q tokens to the present token.*

Proof. We construct the input prompt with positional embedding as follows:

$$\begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \\ \boldsymbol{p}_0 & \boldsymbol{p}_1 & \boldsymbol{p}_2 & \cdots & \boldsymbol{p}_N \\ \boldsymbol{p}_{-1} & \boldsymbol{p}_0 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{p}_{-q} & \boldsymbol{p}_{1-q} & \boldsymbol{p}_{2-q} & \cdots & \boldsymbol{p}_{N-q} \\ 0 & 1 & 2 & \cdots & N \end{bmatrix}.$$

The i-th head aims to connect and communicate the current token with the previous i-th token. Specifically, we let

$$\boldsymbol{W}_K^{(i)}\boldsymbol{P} = \left[\begin{array}{ccccc} \boldsymbol{p}_0 & \boldsymbol{p}_1 & \boldsymbol{p}_2 & \cdots & \boldsymbol{p}_N \end{array}\right],$$

$$\boldsymbol{W}_Q^{(i)}\boldsymbol{P} = \left[\begin{array}{ccccc} \boldsymbol{p}_{-i} & \boldsymbol{p}_{1-i} & \boldsymbol{p}_{2-i} & \cdots & \boldsymbol{p}_{N-i} \end{array}\right],$$

and

$$\boldsymbol{W}_V^{(i)}\boldsymbol{P} = \left[\begin{array}{ccccc} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \end{array}\right],$$

we have

$$\boldsymbol{W}_V^{(i)}\boldsymbol{P}\cdot\mathrm{softmax}\left(\boldsymbol{P}^\top \boldsymbol{W}_K^{(i)\top}\boldsymbol{W}_Q^{(i)}\boldsymbol{P}\right) \approx \left[\begin{array}{ccccc} * & \boldsymbol{x}_{1-i} & \boldsymbol{x}_{2-i} & \cdots & \boldsymbol{x}_{N-i} \end{array}\right].$$

Here, we use "*" to mask some unimportant token values. Therefore, the q-head attention layer outputs

$$\begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \\ * & \boldsymbol{x}_0 & \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{N-1} \\ * & * & \boldsymbol{x}_0 & \cdots & \boldsymbol{x}_{N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & \boldsymbol{x}_{N-q} \\ 0 & 1 & 2 & \cdots & N \end{bmatrix}.$$

Here, the samples are only supported on $\{\boldsymbol{x}_t\}$ with $0 \le t \le N$. It is common to alternatively define $\boldsymbol{x}_t = \mathbf{0}$ for $t < 0$. Passing through the feed-forward neural network, together with the indicators at the last row, we can filter out the undefined elements, i.e.,

$$\begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \\ \mathbf{0} & \boldsymbol{x}_0 & \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_{N-1} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{x}_0 & \cdots & \boldsymbol{x}_{N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{x}_{N-q} \\ 0 & 1 & 2 & \cdots & N \end{bmatrix}.$$

□

Proof for Theorem 3.2 According to Lemma A.3 and Theorem 3.1, the $(L+2)$-layer pre-transformer $\mathrm{TF}_{\mathrm{pre}}$ is able to do copying and transformation of the representation function. The output after the preprocessing is given by

$$\begin{bmatrix} \boldsymbol{x}_0 & \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \\ \Phi^*\left(\boldsymbol{x}_{1-q:0}\right) & \Phi^*\left(\boldsymbol{x}_{2-q:1}\right) & \Phi^*\left(\boldsymbol{x}_{3-q:2}\right) & \cdots & \Phi^*\left(\boldsymbol{x}_{N+1-q:N}\right) \end{bmatrix},$$

where we denote $\boldsymbol{x}_{i:j}$ as the concatenation $[\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_j]$ for notational simplicity. Similar to the construction in Lemma A.2, a one-layer two-head transformer $\mathrm{TF}_{\mathrm{loop}}$ is capable of implementing gradient descent on the multivariate regression. Finally, the post-transformer $\mathrm{TF}_{\mathrm{post}}$ moves the desired result to the output.

A.3 Proof for Theorem 3.3

According to Lemma 5 in (Li et al., 2023), a seven-layer, two-head pre-transformer $\mathrm{TF}_{\mathrm{pre}}$ is introduced for preprocessing the input CoT sequence. This pre-transformer performs filtering, transforming the input sequence into the structured form given by Equation 7.

$$\begin{bmatrix} \mathbf{0} & \cdots & \boldsymbol{s}_1^{\ell-1} & \mathbf{0} & \cdots & \boldsymbol{s}_2^{\ell-1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \hat{\boldsymbol{s}}^{\ell-1} \\ \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{s}_1^{\ell} & \cdots & \mathbf{0} & \boldsymbol{s}_2^{\ell} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ 0 & \cdots & 1 & 0 & \cdots & 1 & 0 & \cdots & 0 & 0 & \cdots & 1 \\ 0 & \cdots & 0 & 1 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{7}$$

Specifically, it identifies the positional index $\ell-1$ of the last token $\hat{\boldsymbol{s}}^{\ell-1}$, retains only $\boldsymbol{s}_i^{\ell-1}$ and $\boldsymbol{s}_i^{\ell}$ for $1 \leq i \leq N$, and filters out all other irrelevant tokens. In this context, the representation function $\Phi^*(\cdot)$ corresponds to L-layer leaky ReLU MLPs. Notably, the transformation $\boldsymbol{s}i^{\ell} = \sigma\left(\boldsymbol{W}^{\ell}\boldsymbol{s}i^{\ell-1}\right)$ is expressed, where $\boldsymbol{W}^{\ell}$ denotes the weight matrix at the ℓ-th layer, and $\sigma(\cdot)$ represents the leaky ReLU activation function. Given the reversibility and piecewise linearity of the leaky ReLU activation, we can assume, without loss of generality, that $\boldsymbol{s}_i^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{s}_i^{\ell-1}$ in Equation 7. Consequently, the problem is reduced to a multi-variate regression, and a one-layer two-head transformer $\mathrm{TF}_{\mathrm{loop}}$ is demonstrated to effectively implement gradient descent for determining the weight matrix $\boldsymbol{W}^{\ell}$, as shown in Lemma A.2. Subsequently, the post-transformer $\mathrm{TF}_{\mathrm{post}}$ produces the desired result $\sigma\left(\boldsymbol{W}^{\ell}\hat{\boldsymbol{s}}i^{\ell-1}\right)$.

A.4 Proof for Theorem 4.1

In this section, we first show that the one-layer two-head transformer can implement a single step of Newton's method in Equation 4, with the special form of input token vectors. Then, we introduce the pre-transformer, designed to convert general input tokens into the prescribed format conducive to the transformer's operation. Finally, the post-transformer facilitates the extraction of the desired results through additional computations, given that the output from the looped-transformer corresponds to an intermediate product.

Lemma A.4. *A transformer with one layer and two heads is capable of implementing one step of Newton's method in the linear regression problem in Equation 3.*

Proof. Let us consider the input prompt with positional embedding as follows:

$$\boldsymbol{P} := \begin{bmatrix} \boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{x}_N & \mathbf{0} & \boldsymbol{x}_{\text{test}} & \mathbf{0} \\ 0 & y_1 & \cdots & 0 & y_N & 0 & 0 \\ \boldsymbol{M}_k\boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{M}_k\boldsymbol{x}_N & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Let

$$\boldsymbol{W}_Q\boldsymbol{P} = \begin{bmatrix} \boldsymbol{M}_k\boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{M}_k\boldsymbol{x}_N & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & \cdots & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_K\boldsymbol{P} = \begin{bmatrix} c\boldsymbol{x}_1 & \mathbf{0} & \cdots & c\boldsymbol{x}_N & \mathbf{0} & c\boldsymbol{x}_{\text{test}} & \mathbf{0} \\ 0 & 0 & \cdots & 0 & 0 & 0 & C \end{bmatrix},$$

and

$$\boldsymbol{W}_V\boldsymbol{P} = e^C \begin{bmatrix} \boldsymbol{M}_k\boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{M}_k\boldsymbol{x}_N & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}.$$

Similarly, denote $\boldsymbol{Z} := \boldsymbol{P}^\top\boldsymbol{W}_K^\top\boldsymbol{W}_Q\boldsymbol{P}$, we can establish that

$$e^C\mathrm{softmax}\left(Z_{2i-1,2j-1}\right) \approx 1 + c\boldsymbol{x}_i^\top\boldsymbol{M}_k\boldsymbol{x}_j. \tag{8}$$

To nullify the constant term, an additional attention head can be incorporated. Therefore, the output takes the form:

$$\boldsymbol{W}_V \boldsymbol{P}\mathrm{softmax}\left(\boldsymbol{P}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{P}\right) \approx \begin{bmatrix} c\boldsymbol{M}_k\boldsymbol{S}\boldsymbol{M}_k\boldsymbol{x}_1 & * & \cdots & c\boldsymbol{M}_k\boldsymbol{S}\boldsymbol{M}_k\boldsymbol{x}_N & * & * \end{bmatrix}.$$

Here, we use "*" to mask some unimportant token values. Upon passing through the feed-forward neural network with indicator $\begin{bmatrix} 1 & 0 & \cdots & 1 & 0 & 0 & 0 \end{bmatrix}$ and weight $1/c$, the resulting output is

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 0 & y_1 & \cdots & 0 & y_N & 0 & 0 \\ \boldsymbol{M}_{k+1}\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{M}_{k+1}\boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ 1 & 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\boldsymbol{M}_{k+1} = 2\boldsymbol{M}_k - \boldsymbol{M}_k\boldsymbol{S}\boldsymbol{M}_k$. □

Proof for Theorem 4.1. For $\mathrm{TF}_{\mathrm{pre}}$, we adopt the following configurations:

$$\boldsymbol{W}_Q\boldsymbol{P} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ 1 & 1 & \cdots & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_K\boldsymbol{P} = \begin{bmatrix} c\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & c\boldsymbol{x}_N & \boldsymbol{0} & c\boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 0 & 0 & \cdots & 0 & 0 & 0 & C \end{bmatrix},$$

and

$$\boldsymbol{W}_V\boldsymbol{P} = e^C \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}.$$

Denote $\boldsymbol{Z} := \boldsymbol{P}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{P}$, we can show that

$$e^C \mathrm{softmax}\left(Z_{2i-1,2j-1}\right) \approx 1 + c\boldsymbol{x}_i^\top \boldsymbol{x}_j.$$

We may include another attention head to remove the constant. Therefore, the output is formulated as

$$\boldsymbol{W}_V \boldsymbol{P}\mathrm{softmax}\left(\boldsymbol{P}^\top \boldsymbol{W}_K^\top \boldsymbol{W}_Q \boldsymbol{P}\right) \approx \begin{bmatrix} c\boldsymbol{S}\boldsymbol{x}_1 & * & \cdots & c\boldsymbol{S}\boldsymbol{x}_N & * & * \end{bmatrix}.$$

After passing through the feed-forward neural network with indicators $\begin{bmatrix} 1 & 0 & \cdots & 1 & 0 & 0 & 0 \end{bmatrix}$ and weight α/c, the resulting output becomes:

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 0 & y_1 & \cdots & 0 & y_N & 0 & 0 \\ \boldsymbol{M}_0\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{M}_0\boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ 1 & 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\boldsymbol{M}_0 = \alpha\boldsymbol{S}$.

As illustrated in Lemma A.4, after T iterations of the looped transformer $\mathrm{TF}_{\mathrm{loop}}$, it produces the following output:

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{x}_{\text{test}} & \boldsymbol{0} \\ 0 & y_1 & \cdots & 0 & y_N & 0 & 0 \\ \boldsymbol{M}_T\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{M}_T\boldsymbol{x}_N & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ 1 & 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In the post-transformer, additional positional embeddings are introduced to address technical considerations. The input is structured as follows:

$$\begin{bmatrix} \boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{x}_N & \mathbf{0} & \boldsymbol{x}_{\text{test}} & \mathbf{0} \\ 0 & y_1 & \cdots & 0 & y_N & 0 & 0 \\ \boldsymbol{M}_T\boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{M}_T\boldsymbol{x}_N & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1} \\ 1 & 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix},$$

where the positional embedding vectors $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_{N+1}$ are designed to be nearly orthogonal (Lemma A.1). To initiate the weight $\boldsymbol{w}_T^{\text{Newton}}$, we propagate the target label y to adjacent tokens using the following attention mechanism:

$$\boldsymbol{W}_K\boldsymbol{P} = \boldsymbol{W}_Q\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1} \end{bmatrix}$$

and

$$\boldsymbol{W}_V\boldsymbol{P} = 2\begin{bmatrix} 0 & y_1 & \cdots & 0 & y_N & 0 & 0 \end{bmatrix}.$$

This operation results in the attention layer producing the following output:

$$\begin{bmatrix} \boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{x}_N & \mathbf{0} & \boldsymbol{x}_{\text{test}} & \mathbf{0} \\ y_1 & y_1 & \cdots & y_N & y_N & 0 & 0 \\ \boldsymbol{M}_T\boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{M}_T\boldsymbol{x}_N & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{p}_1 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_N & \boldsymbol{p}_N & \boldsymbol{p}_{N+1} & \boldsymbol{p}_{N+1} \\ 1 & 0 & \cdots & 1 & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In the next layer, analogous to the construction in Equation 8, we define the following transformations:

$$\boldsymbol{W}_Q\boldsymbol{P} = \begin{bmatrix} c\boldsymbol{x}_1 & \mathbf{0} & \cdots & c\boldsymbol{x}_N & \mathbf{0} & c\boldsymbol{x}_{\text{test}} & \mathbf{0} \\ 1 & 1 & \cdots & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_K\boldsymbol{P} = \begin{bmatrix} \boldsymbol{M}_T\boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{M}_T\boldsymbol{x}_N & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & \cdots & 0 & 0 & 0 & C \end{bmatrix},$$

and

$$\boldsymbol{W}_V\boldsymbol{P} = e^C \begin{bmatrix} y_1 & y_1 & \cdots & y_N & y_N & 0 & 0 \end{bmatrix},$$

for some $C, c > 0$. Defining the matrix

$$\boldsymbol{Z} := \boldsymbol{P}^\top\boldsymbol{W}_K^\top\boldsymbol{W}_Q\boldsymbol{P} = \begin{bmatrix} c\boldsymbol{x}_1^\top\boldsymbol{M}_T\boldsymbol{x}_1 & 0 & \cdots & c\boldsymbol{x}_1^\top\boldsymbol{M}_T\boldsymbol{x}_N & 0 & c\boldsymbol{x}_1^\top\boldsymbol{M}_T\boldsymbol{x}_{\text{test}} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ c\boldsymbol{x}_N^\top\boldsymbol{M}_T\boldsymbol{x}_1 & 0 & \cdots & c\boldsymbol{x}_N^\top\boldsymbol{M}_T\boldsymbol{x}_N & 0 & c\boldsymbol{x}_N^\top\boldsymbol{M}_T\boldsymbol{x}_{\text{test}} & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ C & C & \cdots & C & C & C & C \end{bmatrix},$$

we can show that

$$e^C \cdot \operatorname{softmax}\left(Z_{2i-1,2j-1}\right) \approx 1 + c\boldsymbol{x}_i^\top\boldsymbol{M}_T\boldsymbol{x}_j,$$

where the closeness of the two sides of the approximation "≈" can be achieved by selecting $C > 0$ sufficiently large and $c > 0$ sufficiently small. The constant term can be removed by introducing another head. Therefore, the output of the attention layer is expressed as

$$\sum_{i=1}^{2} \boldsymbol{W}_V^{(i)}\boldsymbol{X} \cdot \operatorname{softmax}\left(\boldsymbol{X}^\top\boldsymbol{W}_K^{(i)\top}\boldsymbol{W}_Q^{(i)}\boldsymbol{X}\right) \approx c\begin{bmatrix} * & * & \cdots & * & * & \boldsymbol{w}_T^{\text{Newton}\top}\boldsymbol{x}_{\text{test}} & * \end{bmatrix}.$$

Finally, the feed-forward neural network yields the desired prediction $\boldsymbol{w}_T^{\text{Newton}\top}\boldsymbol{x}_{\text{test}}$.

A.5 Proof for Theorem 4.2

In this section, we extend the realization of gradient descent, as demonstrated in Lemma A.2 for encoder-based transformers, to decoder-based transformers. Although the construction is similar, the key distinction lies in the decoder-based transformer's utilization of previously viewed data for regression, consistent with our intuitive understanding. The following lemma is enough to conclude the proof for Theorem 4.2.

Lemma A.5. *The one-layer two-head decoder-based transformer can implement one step of gradient descent in linear regression problems in Equation 5.*

Proof. We consider the input prompt with positional embedding as follows:

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_{i-1} & \boldsymbol{0} & \boldsymbol{x}_i \\ 0 & y_1 & \cdots & 0 & y_{i-1} & 0 \\ \boldsymbol{0} & \boldsymbol{x}_1 & \cdots & \boldsymbol{0} & \boldsymbol{x}_{i-1} & \boldsymbol{0} \\ \boldsymbol{w}_k^1 & \boldsymbol{0} & \cdots & \boldsymbol{w}_k^{i-1} & \boldsymbol{0} & \boldsymbol{w}_k^i \\ 0 & 0 & \cdots & \frac{1}{i-2} & 0 & \frac{1}{i-1} \\ 1 & 0 & \cdots & 1 & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

We construct the attention layer with

$$\boldsymbol{W}_Q\boldsymbol{P}^i = \begin{bmatrix} \boldsymbol{w}_k^1 & \boldsymbol{0} & \cdots & \boldsymbol{w}_k^{i-1} & \boldsymbol{0} & \boldsymbol{w}_k^i \\ 1 & 1 & \cdots & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_K\boldsymbol{P}^i = \begin{bmatrix} c\boldsymbol{x}_1 & \boldsymbol{0} & \cdots & c\boldsymbol{x}_{i-1} & \boldsymbol{0} & c\boldsymbol{x}_i \\ 0 & C & \cdots & 0 & 0 & 0 \end{bmatrix},$$

and

$$\boldsymbol{W}_V\boldsymbol{P}^i = e^C/c \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_{i-1} & \boldsymbol{0} & \boldsymbol{x}_i \end{bmatrix}.$$

Here, we adopt causal attention, where the attention mechanism can only attend to previous tokens. The output is

$$\begin{bmatrix} * & * & \cdots & \sum_{j=1}^{i-2} \boldsymbol{w}_k^{i-1\top}\boldsymbol{x}_j\boldsymbol{x}_j & * & \sum_{j=1}^{i-1} \boldsymbol{w}_k^{i\top}\boldsymbol{x}_j\boldsymbol{x}_j \end{bmatrix}.$$

For the second head, we similarly let

$$\boldsymbol{W}_Q\boldsymbol{P}^i = \begin{bmatrix} 1 & 0 & \cdots & 1 & 0 & 1 \\ 1 & 0 & \cdots & 1 & 0 & 1 \end{bmatrix},$$

$$\boldsymbol{W}_K\boldsymbol{P}^i = \begin{bmatrix} 0 & cy_1 & \cdots & 0 & cy_{i-1} & 0 \\ C & 0 & \cdots & 0 & 0 & 0 \end{bmatrix},$$

and

$$\boldsymbol{W}_V\boldsymbol{P}^i = -e^C/c \begin{bmatrix} \boldsymbol{0} & \boldsymbol{x}_1 & \cdots & \boldsymbol{0} & \boldsymbol{x}_{i-1} & \boldsymbol{0} \end{bmatrix}.$$

Then, we have the output

$$\begin{bmatrix} * & * & \cdots & -\sum_{j=1}^{i-2} y_j\boldsymbol{x}_j & * & -\sum_{j=1}^{i-1} y_j\boldsymbol{x}_j \end{bmatrix}$$

The attention layer outputs

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{0} & \cdots & \boldsymbol{x}_{i-1} & \boldsymbol{0} & \boldsymbol{x}_i \\ 0 & y_1 & \cdots & 0 & y_{i-1} & 0 \\ \boldsymbol{0} & \boldsymbol{x}_1 & \cdots & \boldsymbol{0} & \boldsymbol{x}_{i-1} & \boldsymbol{0} \\ \boldsymbol{w}_k^1 & \boldsymbol{0} & \cdots & \boldsymbol{w}_k^{i-1} & \boldsymbol{0} & \boldsymbol{w}_k^i \\ * & * & \cdots & \sum_{j=1}^{i-2}\left(\boldsymbol{w}_k^{i-1\top}\boldsymbol{x}_j - y_j\right)\boldsymbol{x}_j & * & \sum_{j=1}^{i-1}\left(\boldsymbol{w}_k^{i\top}\boldsymbol{x}_j - y_j\right)\boldsymbol{x}_j \\ 0 & 0 & \cdots & \frac{1}{i-2} & 0 & \frac{1}{i-1} \\ 1 & 0 & \cdots & 1 & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \end{bmatrix}.$$

Since the feed-forward layer is capable of approximating nonlinear functions, e.g., multiplication, the transformer layer outputs

$$\begin{bmatrix} \boldsymbol{x}_1 & \mathbf{0} & \cdots & \boldsymbol{x}_{i-1} & \mathbf{0} & \boldsymbol{x}_i \\ 0 & y_1 & \cdots & 0 & y_{i-1} & 0 \\ \mathbf{0} & \boldsymbol{x}_1 & \cdots & \mathbf{0} & \boldsymbol{x}_{i-1} & \mathbf{0} \\ \boldsymbol{w}_{k+1}^1 & \mathbf{0} & \cdots & \boldsymbol{w}_{k+1}^{i-1} & \mathbf{0} & \boldsymbol{w}_{k+1}^{i} \\ 0 & 0 & \cdots & \frac{1}{i-2} & 0 & \frac{1}{i-1} \\ 1 & 0 & \cdots & 1 & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \end{bmatrix},$$

where $\boldsymbol{w}_{k+1}^j = \boldsymbol{w}_k^j - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} \left(\boldsymbol{w}_k^j; \boldsymbol{P}^j \right) = \boldsymbol{w}_k^j - \frac{\eta}{j-1} \sum_{h=1}^{j-1} \left(\boldsymbol{w}_k^{h\top} \boldsymbol{x}_h - y_h \right) \boldsymbol{x}_h$. □