Chain-of-Thought Gradient Descent

Anonymous Authors[1]

Abstract

We show that Chain-of-Thought (CoT) enables a fixed single-layer transformer to efficiently approximate the training process of an N-layer feed-forward network in-context. Since FFNs are universal approximators, this result provides strong theoretical evidence for the expressive power of CoT. Specifically, we improve the computational cost of the prior best in-context result [Wu et al., ICML 2025] by $O(N)$. Building on the insight of the recursive nature of CoT, we reuse the single-layer transformer autoregressively instead of stacking the same transformer blocks to perform multiple In-Context Gradient Descent (ICGD) updates. The key novelty is a dynamic-masking scheme: at each CoT step, the attention heads are forced to see only the tokens needed to compute the result of the current forward or backpropagation update. This selective reuse contrasts with earlier ICGD proofs for neural network optimization, which must carry all information through every layer simply because a later gradient update might need it. Our numerical validations backup our theory.

1 Introduction

We study the CoT's ability to simulate the multi-step gradient descent of deep learning and statistical models in-context. We prove that a one-layer transformer block, equipped with Chain-of-Thought (CoT) and dynamic masking, carry out L steps of gradient descent on an N-layer network with $O(1)$ depth and an $O(N)$ compute saving over prior work (Wu et al., 2025). The motivation for investigating CoT to simulate In-Context Gradient Descent (ICGD) of deep NN is from an observation that the model size of a frozen transformer that can learn through ICGD is restricted by the input size. We first introduce the importance of ICGD and later explain its restriction in the next paragraph.

Several works explain the in-context learning ability of transformer through the lens of ICGD (Akyürek et al., 2022; Dai et al., 2022; Cheng et al., 2023; Von Oswald et al., 2023; Bai et al., 2024; Wang et al., 2024; Wu et al., 2025). Given a few in-context examples, a frozen transformer simulates the multi-step gradient descent of different machine learning models by propagating input down the deep transformer. This theoretical result is also backed up by empirical analysis (Garg et al., 2022). However, in the prior works when proving transformer simulates a complex or larger machine learning model (such as N-layer NN) in-context (Wang et al., 2024; Wu et al., 2025), the input scales with the model we want to simulate. This is because input is the only place a frozen transformer can save the model information we aim to simulate. Specifically, they place all N-layer network weights and intermediate activations into the embedding of every token. This causes significant redundant computation, since in their construction, each transformer layer executes either a forward or backward update and needs only small slices of embeddings. However, the self-attention and MLP blocks are forced to process the entire embedding vector. Since the embedding dimension d must scale with the simulated network depth N, this effectively multiplies the matrix-vector multiplication cost by $O(N)$ for every operation compared to processing only the needed part of embedding. Additionally, since the input size is fixed in these in-context learning settings (unlike CoT), this imposes constraints on the complexity of machine learning algorithms transformer can simulate in-context.

Our insight to resolve this is CoT with dynamic masking. First, we note that if the prompt evolves as a CoT, just using a one-layer transformer blocks recursively is able to simulate the training dynamics of the same N-th layer neural network previously required $(2N + 4)L$-layer transformer. Furthermore, when equipped with dynamic masking, we force each attention head at every step to read only the entries needed for the current forward or backward update, letting other data stay hidden until required. By this, we resolve the issue of propagating redundant information into the current forward or backpropagation computation of transformer layer. Yang et al. (2025) investigate similar idea through introducing a reduction rule to delete token spans once they become irrelevant, and demonstrate it through experiments.

Contributions. Our contribution are three-folds:

- **CoT-driven ICGD.** We prove that one-layer transformer blocks with CoT simulate arbitrary step in-context gradient descent of N-layer neural network.

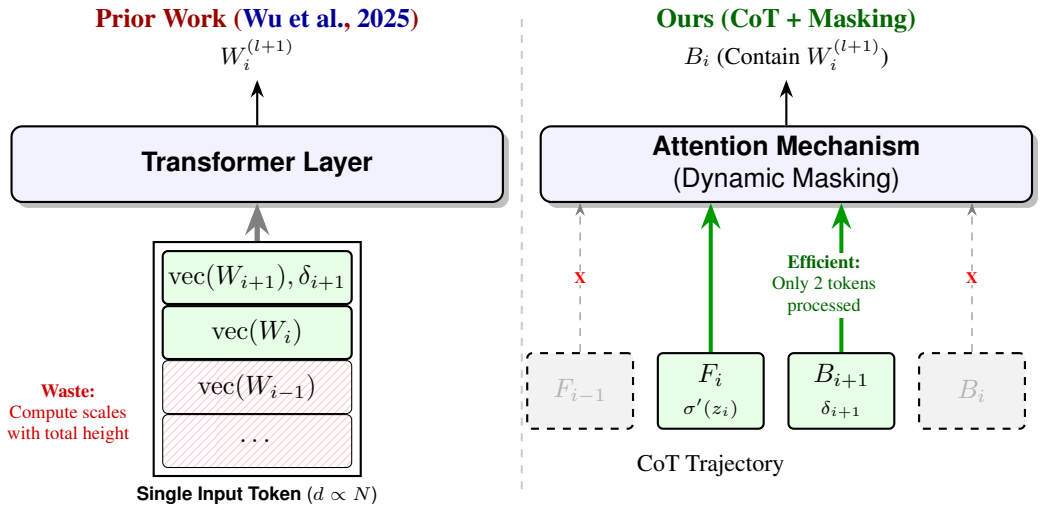

Figure 1. **Comparison of Computational Cost.** We demonstrate the concept on one-step backpropogation on i-th layer of neural network. **Left:** Wu et al. (2025) stack all weights $\text{vec}(W_{1:N})$ vertically into a single high-dimensional input vector, forcing the transformer to process the full stack every step. **Right:** Our approach keeps weights in separate tokens (F, B). Dynamic Masking (Green Arrows) retrieves only the relevant tokens (F_i, B_{i+1}), keeping the computation constant $O(1)$. We omit the whole CoT trajectory for simplicity.

We provide explicit construction of such transformer block in Section 4.

- **Dynamic masking.** We identify a key drawback of ICGD: each layer processes redundant information because a deeper transformer layer requires it. We apply dynamic masking to reduce the total computation of the SOTA result (Wu et al., 2025) on simulating training N-layer neural network by $O(N)$. Our CoT with dynamical masking provides a refined architecture design to enhance the transformer's in-context learning ability and efficiency. Compared with the concurrent work (Yang et al., 2025), we provide another theoretical perspective to explain the success of this technique.
- **Extension on Various Statistical Models.** We provide various approximation results on the fundamental statistical models. The result demonstrates through CoT, single-layer attention, or single-layer transformer, approximate generalized linear model, ridge regression, and lasso regression (Section 3).

Related Work. Please see Section B for the detailed discussion of the related work.

Organization. Section 2 provides background knowledge and CoT setting considered in this paper. Section 3 provides examples of how CoT recursive nature saves $O(L)$ memory requirement for transformer simulating ICGD on statistical models. Section 4 presents our main result, with explicit construction of a shallow transformer equipped with dynamic masking to perform arbitrary stepa of ICGD simulation on N-layer neural network. We include a detailed discussion of related work in Section B. We place all the proofs in Section C, and the numerical experiment is in Section D.

2 Preliminaries and Problem Setup

In this section, we establish the formal setup of the Chain-of-Thought In-Context Gradient Descent (CoT-ICGD) framework we analyze in this paper. To provide enough background, We introduce the concepts of (i) in-context learning, (ii) in-context gradient descent, (iii) CoT-ICGD setup, and finally (iv) gradient-descent computations for deep neural networks.

In-Context Learning. We follow the notation from Wu et al. (2025) in this section. Consider a dataset $\mathcal{D}_n := \{(x_i, y_i)\}_{i\in[n]}$ drawn i.i.d. from a distribution $\mathbb{P}$, where $\{x_i\}_{i\in[n]} \subseteq \mathbb{R}^d$ is the input of f and $\{y_i\}_{i\in[n]} \subseteq \mathbb{R}$ is the function output. Importantly, each in-context example $(\mathcal{D}_n, x_{n+1})$ comes from a different distribution $\mathbb{P}_j$, such as a different ground truth linear model or neural network parametrized by $w_j^\star \in \mathbb{R}^d$. The in-context learning ability refers to a pretrained transformer that can predict the output corresponding to a test input x_{n+1} based on the in-context examples $\mathcal{D}_n$ without parameter updates across many P_j. Formally, we encode the context and test input $(\mathcal{D}_n, x_{n+1})$ as a prompt $Z \in \mathbb{R}^{D\times(n+1)}$

$$Z = \begin{bmatrix} x_1 & x_2 & \cdots & x_n & x_{n+1} \\ y_1 & y_2 & \cdots & y_n & 0 \\ p_1 & p_2 & \cdots & p_n & p_{n+1} \end{bmatrix} \in \mathbb{R}^{D\times(n+1)}, \quad (1)$$

where each vector $p_i \in \mathbb{R}^{D\times(n+1)}$ holds any necessary auxiliary information such as positional encoding.

In-Context Gradient Descent. To define what we mean by a transformer implementing in-context gradient descent, we consider a parametric model $f(w, \cdot) : \mathbb{R}^{D_w} \to \mathbb{R}$, we define the empirical risk on the in-context examples $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$ as

$$\mathcal{L}_n(w) = \frac{1}{2n} \sum_{i=1}^{n} \ell(f(w, x_i), y_i).$$

We say the transformer $\mathrm{TF}(\cdot)$ *implements in-context gradient descent* if, given the current token representation

$$z_i^{(l)} = [x_i;\, y_i;\, \bar{w}^{(l)};\, p_i],$$

passing it through the transformer updates it to

$$\mathrm{TF}(z_i^{(l)}) = z_i^{(l+1)} = [x_i; y_i; \bar{w}^{(l+1)}; p_i],$$

by approximating one gradient descent step

$$\bar{w}^{(l+1)} = \bar{w} - \eta g^{(l)} + \varepsilon^{(l)},$$
$$g^{(l)} = \frac{1}{|B_l|} \sum_{i \in B_l} \nabla \ell(f(w, x_i), y_i),$$

where $\varepsilon^{(l)}$ denotes the approximation error at step l. Here $B_l \subseteq \{1, \ldots, n\}$ is the set of sample indices used to compute the gradient at step l. Taking $B_l = \{1, \ldots, n\}$ recovers gradient descent, while any $|B_l| = m < n$ yields stochastic gradient descent (in our NN setting in Section 4 we take $m = 1$).

Then we move to the core setting of this work. To implement multiple gradient-descent steps within a fixed-size transformer, we introduce the Chain-of-Thought In-Context Gradient Descent Setup.

Definition 2.1 (Chain-of-Thought In-Context Gradient Descent Setup)**.** *The transformer maintains a structured CoT format. At each generation step t, the prompt Z_t equals*

$$Z_t = \begin{bmatrix} G_1 & G_2 & \cdots & G_t \end{bmatrix} \in \mathbb{R}^{d \times tn}.$$

Here, each block $G_t \in \mathbb{R}^{d \times n}$ contains the information necessary to perform the next step in the CoT gradient-descent procedure. Depending on the scenario, this may include the current parameter update, data samples, intermediate activations (in forward propagation), gradients (in backward propagation), and other auxiliary information. At iteration t, given the current input Z_t, the transformer $\mathrm{TF} : \mathbb{R}^{d \times tn} \to \mathbb{R}^{d \times n}$ generate the next block G_{t+1}

$$G_{t+1} = \mathrm{TF}(Z_t).$$

The prompt then grows autoregressively as

$$Z_{t+1} = \begin{bmatrix} Z_t & G_{t+1} \end{bmatrix} \in \mathbb{R}^{d \times (t+1)n}.$$

Remark 2.1 (Multiple-Token Prediction)**.** *Our transformer generates an n-token block per step, following the multi-token-prediction strategy common in modern LLM architectures (Gloeckle et al., 2024; Liu et al., 2024).*

Remark 2.2 (Explicit Form of G_t.)**.** *For statistical model (Section 3), we use subscript l instead of t to to indicate it contains the l-th gradient update. Each block G_l stores $[x_i; y_i; w^{(l)}; 1]_{i \in [n]}$. For deep networks, blocks alternate between forward-pass block F_i and backward-pass states B_i for $i \in [N]$, see Section 4 for the details.*

Finally, to prepare for subsequent theoretical analysis (Section 4), we recall the standard definitions of forward and backward computations for gradient descent in multi-layer neural networks.

N-Layer Neural Network Gradient Descent. We define the forward computation in Definition 2.2 and backpropagation in Definition 2.3. Later in Section 4, we replicate the forward and backward computations from (2)–(4) through a CoT mechanism.

Definition 2.2 (N-Layer Feed-Forward Layer)**.** *Let $x \in \mathbb{R}^d$ be the input, and let $W_i \in \mathbb{R}^{d \times d}$ and $b \in \mathbb{R}^d$ denote the weights and bias of the i-th layer for $i \in [N]$. Denote the activation function as $\sigma(\cdot)$. Define pre-activation z_i and activation o_i as*

$$z_i := W_i o_{i-1} + b_i, \quad o_i := \sigma(z_i),$$

where we set input $o_0 = x$, and note that o_N is the output of FFN.

Definition 2.3 (Backpropagation of Gradient Update for N-layer FFN)**.** *Given target label y, denote the loss function as $\ell := \ell(o_N, y)$. The backpropagation computes the gradient in a recursive manner as follows. First, compute the error at the output layer:*

$$\delta_N = \nabla_{o_N} \ell \odot \sigma_N'(z_N). \tag{2}$$

Second, compute $\delta_{N-1}, \cdots, \delta_1$ by propogating the error backwards. For $i = N-1, N-2, \ldots, 1$, propagate the error backwards:

$$\delta_i = \underbrace{W_{i+1}^\top \delta_{i+1}}_{d \times 1} \odot \underbrace{\sigma_i'(z_i)}_{d \times 1}. \tag{3}$$

Third, compute gradients with respect to weights and biases:

$$\frac{\partial \ell}{\partial W_i} = \delta_i o_{i-1}^\top, \quad \frac{\partial \ell}{\partial b_i} = \delta_i. \tag{4}$$

Notations. We use lower case letters denote (column) vectors and upper case letters denote matrices. let n denote the number of in-context examples and d the feature dimension. For an N-layer feed-forward network, we let $W_1, \ldots, W_N$, each in $\mathbb{R}^{d \times d}$, denote the layer weights. For statistical model, we denote $w^{(l)} \in \mathbb{R}^d$ after the l-th gradi-

ent update, where $l = 1, \ldots, L$ and L is the total number of ICGD updates. We use $0_d \in \mathbb{R}^d$ to denote the zero column vector and use $1_d \in \mathbb{R}^d$ to to denote the one column vector. We also use $e_i^{(d)} \in \mathbb{R}^d$ to denote one-hot vector whose i-th is the only non-zero entry and is equal to 1.

3 CoT Enables Single-Layer ICGD of Statistical Models

In this section, we demonstrate a series of theoretical approximations power of CoT prompting in the ICGD fashion. We prove that by only *single-layer* attention (Theorem 3.2 and Theorem 3.1) or *single-transformer block* (Theorem 3.3), equipped with CoT prompt, the model learn multiple fundamental statistical models through in-context gradient descent. This improve prior works (Bai et al., 2023) by $O(L)$ memory consumption, and improve the unrealistic ReLU attention into softmax attention. The statistical model includes ridge regression, generalize linear model, and lass regression.

Each proof in this section reuses the same prompt format defined below.

Definition 3.1 (CoT Prompt Format for Statistical Model)**.** *At iteration $l \in [L]$, the model sees $Z_l \in \mathbb{R}^{(2d+3)\times ln}$ in the form of*

$$Z_l = \begin{bmatrix} G_1 & \cdots & G_l \end{bmatrix} = \begin{bmatrix} x_1 & \ldots & x_n & \cdots & x_1 & \ldots & x_n \\ y_1 & \ldots & y_n & \cdots & y_1 & \ldots & y_n \\ w^{(1)} & \ldots & w^{(1)} & \cdots & w^{(l)} & \ldots & w^{(l)} \\ 1 & \ldots & 1 & \cdots & 1 & \ldots & 1 \end{bmatrix},$$

where each block G_l with n columns stores one gradient-descent update.

Considering the model already finish l-th update, every proof in this section follows two steps.

1. **Select** the most recent block $G_l \in \mathbb{R}^{d\times n}$ by a n length context windows.
2. **Produce** next gradient update G_l with one multi-head attention layer plus a token-wise linear map. we extend the (Hu et al., 2025, Theorem A.1, In-Context Gradient Descent) to the CoT setting. According to the universal approximation power of softmax attention, we approximate gradient $\nabla_w \mathcal{L}(w)$ for any C^1 loss.

Softmax Attention ICGD Approximation. Before diving into specific statistical models, we state a foundational result that a single-layer softmax attention equipped with CoT is a universal approximator for gradient descent updates. We state the ICGD theorem in the CoT setting as follows.

Lemma 3.1 (In-Context Gradient Descent, Theorem 4.1 of (Hu et al., 2025))**.** *Let $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be any C^1 loss function defined on $(w^\top x_i, y_i)$. With input X_l in the form of*

$$X := \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ w^{(l)} & w^{(l)} & \cdots & w^{(l)} \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

when X is bounded, there exists a multi-head self-attention Attn_m with skip connections and each attached with a linear layer A, such that for any $\epsilon > 0$, irrelevant of X, we have:

$$\|\mathrm{Attn}_\mathrm{m} \circ A(X) - \begin{bmatrix} x_{1:n} \\ y_{1:n} \\ (w^{(l)} - \eta \nabla_w \mathcal{L}(w^{(l)}))1_n^\top \\ 1_n^\top \end{bmatrix} \|_\infty \le \epsilon,$$

where $x_{1:n} = [x_1, ..., x_n], y_{1:n} = [y_1, ..., y_n]$, η denotes the learning rate and $\mathcal{L}(w) := \sum_{i=1}^n \ell(w^\top x_i, y_i)$ is an empirical loss upon the given input-output pairs.

Remark 3.1. *Lemma 3.1 comes from (Hu et al., 2025)[Theorem 4.1]. establishes that a single-layer softmax attention approximate one-step gradient descent update for any C^1 loss.*

We now demonstrate how to leverage this single-step approximation capability within our CoT framework (Definition 2.1) to perform multi-step ICGD for specific statistical models. We begin with Generalized Linear Models, a broad and fundamental class of models whose learning often involves minimizing such C^1 loss functions.

3.1 Generalized Linear Model

Consider data $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Define the GLM estimator w_GLM as

$$w_\mathrm{GLM} \coloneqq \operatorname*{argmin}_{w\in\mathbb{R}^d} \mathcal{L}_\mathrm{GLM}(w) = \operatorname*{argmin}_{w\in\mathbb{R}^d} (\sum_{i=1}^n \ell_\mathrm{GLM}(w^\top x_i, y_i)),$$

with $\ell_\mathrm{GLM}(s, y) = -ys + \int_0^s g(t)\, dt$, where the link function $g : \mathbb{R} \to \mathbb{R}$ is continuous. Then we define a ball in ℓ_2-norm for later use.

Definition 3.2. *Let $B_2^k(R)$ denote a ball containing all points in $\mathbb{R}^k$ whose ℓ_2-norm is at most R.*

Then we have the following theorem.

Theorem 3.1 (CoT ICGD Approximation of Generalized Linear Model)**.** *Assume that L_GLM is α-strongly convex and β-smooth on the ball $\mathbb{B}_2(B_w)$, where B_w satisfies $\|w^*_\mathrm{GLM}\|_2 \le B_w/2$, and $w^*_\mathrm{GLM} = \arg\min_w \mathcal{L}_\mathrm{GLM}(w)$. Under this setting, there exists a multi-head attention mechanism Attn_m with a linear mapping A and a feedforward*

network FFN_1 *such that at each step* l*, the transformer approximates a single gradient update*

$$\left\|\mathrm{Attn}_m \circ A(Z_l) - \begin{bmatrix} x_{1:n} \\ y_{1:n} \\ (w^{(l)} - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w))\, 1_n^\top \\ 1_n^\top \end{bmatrix}\right\|_\infty \le \epsilon.$$

where $x_{1:n} = [x_1, ..., x_n], y_{1:n} = [y_1, ..., y_n]$.

For $\eta \le 2/\beta$ *and* $L \le B_w/(2\epsilon)$*, after* L *iterations, the final CoT weight* $w_{\mathrm{CoT}}^{(L)}$ *approximates the true minimizer* w_{GLM} *with the error bound*

$$\|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{GLM}}\|_2 \le L\epsilon + e^{-L/2\kappa}\frac{B_w}{2}.$$

Proof Sketch. We first use continuity of $\partial_s \ell(s, y) = -y + g(s)$ and Lemma 3.1 to show that, a single-layer multi-head attention block with context windows of size n approximates a gradient descent step within error ϵ. Reusing the above attention blocks generate the next predictions in the sense of Definition 2.1, and we use Lemma A.2 to accumulate the each step error to $L\epsilon$. Finally, by classical result the gradient descent has exponential convergence rate toward w_{GLM} under strong convexity and smoothness (Lemma A.1). Combine the both error term through the triangle inequality yields the final result. See Section C.1 for a detailed proof. □

Remark 3.2. *For simplicity we drop the final in-context prediction* x_{n+1}*. This prediction is trivial by adding a* 2*-head attention layer constructed in (Bai et al., 2024).*

We now show the same single-layer attention achieve same multi-step ICGD result on ridge regression. It is a linear model employing ℓ_2 regularization, and its common technique o prevent overfitting and a standard benchmark in studies of in-context learning (Akyürek et al., 2022; Bai et al., 2024).

3.2 Ridge Regression

We first state the ridge regression formally as follows. Consider the ridge regression estimators $w_{\mathrm{ridge}}^{\lambda}$ over the in-context examples in $\mathcal{D}$ with regularization $\lambda \ge 0$:

$$w_{\mathrm{ridge}}^{\lambda} \coloneqq \operatorname*{argmin}_{w \in \mathbb{R}^d} \mathcal{L}_{\mathrm{ridge}}(w)$$
$$= \operatorname*{argmin}_{w \in \mathbb{R}^d} \left(\frac{1}{2n}\sum_{i=1}^{n}(w^\top x_i - y_i)^2 + \frac{\lambda}{2}\|w\|_2^2\right).$$

We have the following results.

Theorem 3.2 (Chain-of-Thought ICGD Approximation of Ridge Regression)**.** *Fix* $\lambda \ge 0$ *and assume* $\|w_{\mathrm{ridge}}^{\lambda}\|_2 \le B_w/2$*. Let* $\eta \le 2/\beta$ *and choose the CoT length* $L \le B_w/(2\epsilon)$*. Then there exists a multi-head attention layer with a linear map such that the CoT iterates satisfy*

$$\|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{ridge}}^{\lambda}\|_2 \le L\epsilon + e^{-L/(2\kappa)}\frac{B_w}{2}.$$

Proof. The proof follwing the same logic as Theorem 3.1. See Section C.2 for a detailed proof. □

Previous two examples focus on smooth loss functions. To demonstrate a broader applicability of our single-layer CoT-ICGD setting, we next extend the result to lasso regression which containing non-smooth ℓ_1 regularization.

3.3 Lasso Regression

We consider approximating proximal gradient descent for lasso regression. Specifically, the lasso minimizer and loss is defined as

$$w_{\mathrm{lasso}} \coloneqq \operatorname*{argmin}_{w \in \mathbb{R}^d} \mathcal{L}_{\mathrm{lasso}}(w)$$
$$= \operatorname*{argmin}_{w \in \mathbb{R}^d} \left(\frac{1}{2n}\sum_{i=1}^{n}(w^\top x_i - y_i)^2 + \lambda\|w\|_1\right).$$

The classical method for minimizing losses with a non-smooth regularizer (such as ℓ_1 norm) is Proximal Gradient Descent (PGD). The single-step PGD updates parameters via standard gradient steps followed by a proximal operator $\mathrm{prox}_{\eta\mathcal{R}}$

$$w_{\mathrm{PGD}}^{t+1} \coloneqq \mathrm{prox}_{\eta\mathcal{R}}(w_{\mathrm{PGD}}^{t} - \eta\nabla\mathcal{L}_n^0(w_{\mathrm{PGD}}^{t})),$$

where $\mathcal{L}_n^0(w) \coloneqq \frac{1}{2n}\sum_{i=1}^{n}(w^\top x_i - y_i)^2$.

Now we state the ICGD result on lasso regression.

Theorem 3.3 (Chain-of-Thought Proximal ICGD Approximation of Lasso Regression)**.** *Given* $\mathcal{L}_n^0(w) \coloneqq 1/n\sum_{i=1}^{n}(w^\top x_i - y_i)^2$ *and the proximal operator* $\mathrm{prox}_{\eta\lambda\|\cdot\|_1}$ *associated with the lasso regularizer, there exists a Transformer layer consisting of a multi-head attention module* $\mathrm{Attn_m}$*, linear transform* A*, and one FFN layers* FFN*, that approximates a single proximal gradient descent (PGD) step in* (10) *for Lasso regression to arbitrary precision. Specifically, for any* $\epsilon \ge 0$*, we have*

$$\|\mathrm{FFN}_2 \circ \mathrm{Attn_m} \circ A(Z_l) - \begin{bmatrix} x_{1:n} \\ y_{1:n} \\ \mathrm{prox}_{\eta\lambda\|\cdot\|_1}\left(w_{\mathrm{PGD}}^{(l)} - \eta\nabla\mathcal{L}_n^0(w_{\mathrm{PGD}}^{(l)})\right) 1_{1\times n} \\ (l+1)\, 1_{1\times n} \\ 1_{1\times n} \end{bmatrix}\|_\infty \le \epsilon,$$

where $x_{1:n} = [x_1, ..., x_n], y_{1:n} = [y_1, ..., y_n]$*. For* $\eta \le 2/\beta$ *and* $L \le R/(2\epsilon)$*, after* L *iterations, the final CoT weight* $w_{\mathrm{CoT}}^{(L)}$ *approximates the true minimizer* w_{lasso} *in terms of loss function*

$$\mathcal{L}_{\mathrm{lasso}}(w_{\mathrm{CoT}}^{(L)}) - \mathcal{L}_{\mathrm{lasso}}(w_{\mathrm{lasso}})$$

$$\leq (C_0 + \beta R + \lambda\sqrt{d})L\epsilon + \frac{\beta}{2}(L\epsilon)^2 + \frac{\beta R^2}{8L}.$$

Proof Sketch. The proof consist of two steps. First we approximate the standard gradient descent step of least square loss, and second the the proximal operator. We approximate the gradient descent step by multihead attention according to Lemma 3.1. For the proximal operator, we rely on (Bai et al., 2023, Proposition C.1) that the proximal operator for the ℓ_1-norm regularizer is exactly approximable by FFN layer. See Section C.3 for a detailed proof. □

Remark 3.3. *Bai et al. (2024) use the ReLU attention, hence exactly construct the gradient of least-square loss $\partial_s \ell(s,t) = \partial_s \ell(w^T x_i, y_i) = \partial_s \frac{1}{2}(w^T x_i - y_i)^2 = (w^T x_i - y_i)x_i$. Replacing the ReLU attention with softmax attention inevitably introduces an approximation error ϵ at each PGD step. With sufficiently large attention width so that $\epsilon = O(1/L^2)$, we recover the standard convergence rate $O(1/L)$ of proximal gradient descent.*

4 CoT Gradient Descent Simulation of N-Layer Neural Network

In this section, we show CoT is capable of simulating more complex model in-context, the multi-step gradient descent of N-Layer Neural Network (Definition 2.2). We achieve this by a single-layer transformer equipped with a dynamic masking scheme. Importantly, the dynamic masking ensures the transformer attends only to the minimal, necessary information from the CoT history at each step. The proofing idea is to replicate the forward and backward computations (Definition 2.3) through attention and feed-forward layer. We provide explicit construction in Theorem 4.1 and Theorem 4.2. The final result shows that CoT reasoning handle non-convex optimisation of deep networks in context with lower resource requirements.

4.1 Simulating Forward and Backpropagation via CoT

We construct the L-step ICGD simulation of N-layer neural network as a CoT sequence of $2LN$ blocks. To begin with, we define some terminology for later proof.

Definition 4.1. *For any $W \in \mathbb{R}^{k\times d}$, define $\mathrm{vec}(\cdot)$ such that*

$$\mathrm{vec}(X) \in \mathbb{R}^{kd}, \quad (\mathrm{vec}(X))_{(j-1)\,k+i} = W_{i,j},$$

for all $i \in [k], j \in [d]$. That is, $\mathrm{vec}(W)$ stacks the d columns of W, each of length k, into a single kd-dimensional column vector.

Definition 4.2 (Rounds, Phases)**.** *Our in-context training process of N-layer feed-forward network consists of l repeated rounds, each round consisiting of 2 phases—the forward propagation phase and the backward propagation phase. Every $2N$ time steps consists a round, where the first N steps belongs to forward propagation and last N steps belong to backward propagation.*

We now define key block types used to store the state of gradient descent of the i-th layer of N-layer neural network as follow.

Definition 4.3 (Form of Inputs and Outputs)**.** *Let l denote the current round. Denote W_i as the weight of the i-th layer of the target N-layer FFN in the last round. The initial data blocks are in the form of*

$$X_l := \begin{bmatrix} x_l \cdot 1_{1\times d} \\ 0_{(3d+d^2)\times d} \\ 2 \cdot 1_{1\times d} \end{bmatrix}, \quad Y_l := \begin{bmatrix} y_l \cdot 1_{1\times d} \\ 0_{(2d+d^2)\times d} \\ 2_{1\times d} \end{bmatrix},$$

for all $l \in [L]$. The transformer generates the forward block F_i and backward block B_i defined as

$$F_i := \underbrace{\begin{bmatrix} o_i \cdot 1_{1\times d} \\ o_{i-1} \cdot 1_{1\times d} \\ \sigma'(z_i) \cdot 1_{1\times d} \\ \mathrm{vec}(W_i) \cdot 1_{1\times d} \\ 1_{1\times d} \end{bmatrix}}_{(1+3d+d^2)\times d}, B_i := \underbrace{\begin{bmatrix} \delta_i \cdot 1_{1\times d} \\ W_i^\top \delta_i \cdot 1_{1\times d} \\ (\mathrm{vec}(W_i) - \eta \mathrm{vec}(\delta_i o_i^\top)) \cdot 1_{1\times d} \\ 0_{(d+1)\times d\cdot} \end{bmatrix}}_{(1+3d+d^2)\times d}$$

In the first round $l = 1$, the B_i has the special form

$$B_i := \underbrace{\begin{bmatrix} 0_{2d\times d} \\ \mathrm{vec}(W_i) \cdot 1_{1\times d} \\ 0_{(d+1)\times d} \end{bmatrix}}_{(1+3d+d^2)\times d}$$

.

Next we introduce the design of the dynamic masking. According to Definition 2.2, to generate forward block F_i we need to retrieve F_{i-1} and B_i. For generating B_i, according to Definition 2.3 we need to retrieve B_{i-1} and F_i. Hence, we design the dynamic masking as follows.

Definition 4.4 (Dynamic Masking)**.** *Let t be the global generation step of the CoT sequence in Definition 2.1. To generate t-th block, the dynamic masking selects a pair of blocks from the prior sequence at the time step*

$$\begin{cases} t-1,\ t-2t \bmod N, & \text{if} \quad t \bmod N \neq 1, \\ t-1,\ \lfloor t/N \rfloor - 1, & \text{if} \quad t \bmod N = 1. \end{cases}$$

Second, since attention does not change the sequence length, we set the dynamic masking to concatenate the multiple blocks it retrieved from the whole input in the column direction.

Remark 4.1 (Provably Resource Efficiency)**.** *Instead of processing all $\{W_i\}_{1\leq i\leq N}$ in every transformer block (Wang et al., 2024; Wu et al., 2025), dynamic masking only propagates 2 weight matrix to the downstream transformer layer. Specifically, in the proof of Wu et al. (2025), they flatten all the weight matrices $\{W_i\}_{1\leq i\leq N}$ and encode onto each*

token. Hence, in the attention mechanism, the weight matrix multiplication is in the scale of $O(N)$. We instead only process 2 weight matrices at a time. This results in $O(N)$ computation saving.

We now move to the construction transformer layer, implementing the forward and backpropagation.

When $t \bmod 2N \in [N]$, the following network performs forward propagation for the N-layer feed-forward network.

Theorem 4.1 (In-Context Calculation of Forward Propagation)**.** *With the input it receives from the dynamic masking mechanism, for any $\epsilon > 0$, there exists a single-layer transformer network $\mathcal{F}_{\text{forward}}$ with positional encoding such that*

$$\left\| \mathcal{F}_{\text{forward}} \left(\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix} \right) - F_i \right\|_\infty \leq \epsilon,$$

where F_i and B_i are the forward and backpropagation blocks defined in Definition 4.3.

Proof Sketch. First, use a linear transformation to extract the needed information o_{i-1} and $\text{vec}(W_i)$, and construct a single attention to approximate o_i. Then construct the second attention to approximate $\sigma'(z_i)$. See Section C.4 for a detailed proof. □

When $t \bmod 2N \in \{N+1, \cdots, 2N\}$, the following network performs backward propagation for the N-layer feed-forward network.

Theorem 4.2 (In-context Calculation of Backward Propagation)**.** *With the input it receives from the dynamic masking mechanism, For any $\epsilon > 0$, there exists a single-layer transformer network $\mathcal{F}_{\text{backprop}}$ with positional encoding such that*

$$\left\| \mathcal{F}_{\text{backprop}} \left(\begin{bmatrix} B_{i+1} \\ F_i \end{bmatrix} \right) - B_i \right\|_\infty \leq \epsilon,$$

where F_i and B_i are defined in Definition 4.3.

Proof Sketch. First, a linear transformation to extract the needed information from the concatenated input B_{i+1} of F_i. Then we construct a FFN_1 to compute

$$\underbrace{W_{i+1}^\top \delta_{i+1} \odot \sigma'(z)}_{\delta_i} \cdot 1_{1\times d}.$$

Third, by constructing a multihead attention layer, we use part of the heads to multiply above δ_i with o_{i-1}, we get $\text{vec}(\delta_i o_{i-1}^\top) \cdot 1_{1\times d}$. Then construct another linear transform to subtract $\text{vec}(W_i)$ from this output. Finally, use another branch of multihead attention to calculate $W_i^\top \delta_i$ and output it as the final output gives us the desired result. See Section C.5 for a detailed proof. □

We now construct a single layer transformer to initialize the first Forward Block as the base case of the recursive construction.

Proposition 4.1 (Initialization of the First Forward Block)**.** *Fix $s \in [n]$ where n is the number of in-context examples, and let X_s, B_1, and F_1 be the blocks defined in Definition 4.3. Then for any $\varepsilon > 0$, there exists a single Transformer block (with residual/skip connections) $\mathcal{F}_x$ such that, for the input obtained by concatenating the two blocks,*

$$Z = \begin{bmatrix} B_1 \\ X_s \end{bmatrix},$$

the output $\mathcal{F}_x(Z)$ approximates the desired forward block F_1 such that

$$\left\| \mathcal{F}_x(Z) - F_1 \right\|_\infty \leq \varepsilon.$$

Proof. See Section C.6 for detailed proof. □

After completing the full forward computation, we construct a corresponding block to initialize the first backward-propogation.

Proposition 4.2 (Initialization the Backward Block)**.** *Fix $s \in [n]$, where n is the number of in-context examples, and let F_N, Y_s, and B_N be the blocks defined in Definition 4.3. Then for any $\epsilon > 0$, there exists a single Transformer block (with residual/skip connections) $\mathcal{F}_y$ such that, for the input obtained by concatenating the two blocks,*

$$Z = \begin{bmatrix} F_N \\ Y_s \end{bmatrix},$$

the output $\mathcal{F}_y(Z)$ approximates the desired backward block B_N such that

$$\left\| \mathcal{F}_y(Z) - B_N \right\|_\infty \leq \epsilon.$$

Proof. See Section C.7 for detailed proof. □

To avoid manually switching between different block updates, we next show that a single Transformer block can automatically implement the correct one step updated for any valid input.

Proposition 4.3 (Universal One-Step Update Block)**.** *Fix $\varepsilon > 0$. There exists a single-layer Transformer block $\text{TF} : \mathbb{R}^{D\times d} \to \mathbb{R}^{D\times d}$ with the following property.*

Let $\mathcal{Z}$ denote the set of valid routed inputs $Z \in \mathbb{R}^{D\times d}$ that arise from the block formats in Definition 4.3, i.e., Z is exactly one of the four types: forward-type, backward-type, x-type, *or* y-type. *Let $M > 0$ be any constant such that $\|Z\|_\infty \leq M/2$ for all $Z \in \mathcal{Z}$.*

Then for every $Z \in \mathcal{Z}$,

$$\|\mathrm{TF}(Z) - T(Z)\|_\infty \le \varepsilon,$$

where T is the corresponding target one-step update map:

$$T(Z) = \begin{cases} T_{\mathrm{fwd}}(Z), & Z \text{ is forward-type}, \\ T_{\mathrm{bwd}}(Z), & Z \text{ is backward-type}, \\ T_x(Z), & Z \text{ is } x\text{-type}, \\ T_y(Z), & Z \text{ is } y\text{-type}. \end{cases}$$

Proof. See Section C.8 for detailed proof. □

Until now we equip ourself all the building blocks to simulate gradient descent of N-layer neural network. Now we formally state the input format and, how to combine those blocks into single transformer and generate the ICGD trajectory of neural network.

Full Input Format. Represent the prompt as a token matrix $Z \in \mathbb{R}^{D \times d(2L+N)}$. The initial context is the block concatenation

$$Z^{(1)} = \begin{bmatrix} X_1 & Y_1 & \cdots & X_L & Y_L & B_1^{(1)} & \cdots & B_N^{(1)} \end{bmatrix},$$

where X_ℓ, Y_ℓ ,and $B_1^{(1)}, \ldots, B_N^{(1)} \in \mathbb{R}^{D \times d}$are defined ion Definition 4.3.

The following lemma displays the gradient update's final trajectory.

Proposition 4.4. *Fix integers $N \ge 1$ and $L \ge 1$, and a labeled sequence $(x_\ell, y_\ell)_{\ell=1}^L$ with input formatting as above. Let $\varepsilon_{\mathrm{C1}}, \varepsilon_{4.1}, \varepsilon_{\mathrm{C2}}, \varepsilon_{4.2} > 0$ be the accuracy parameters from Proposition 4.1,Theorem 4.1, Proposition 4.2, and Theorem 4.2, and let $\varepsilon_{\mathrm{C3}} > 0$ be the routing accuracy parameter from Proposition 4.3. Define*

$$\begin{aligned} \varepsilon_{\mathrm{round}} := \varepsilon_{\mathrm{C_1}} + (N-1)\varepsilon_{4.1} + \varepsilon_{\mathrm{C_2}} \\ + (N-1)\varepsilon_{4.2} + (2N+1)\varepsilon_{\mathrm{C_3}}. \end{aligned}$$

Then there exists a single-layer Transformer block TF *defined in Proposition 4.3, such that it produces state blocks $\{\widehat{B}_i^{(l)}\}_{i\in[N],\, l=1}^{L+1}$ satisfying $\widehat{B}_i^{(1)} = B_i^{(1)}$ and, for every $l \in [L]$,*

$$\max_{i\in[N]} \left\| \widehat{B}_i^{(l+1)} - B_i^{(l+1)} \right\|_\infty \le l\, \varepsilon_{\mathrm{round}}.$$

In particular,

$$\max_{i\in[N]} \left\| \widehat{B}_i^{(L+1)} - B_i^{(L+1)} \right\|_\infty \le L\, \varepsilon_{\mathrm{round}}.$$

Proof Sketch. See Section C.9 for a detailed proof. □

Remark 4.2. *The dynamic masking mechanism (Definition 4.4) ensures that at each of these $2LN$ steps, the fixed-depth transformer only processes a small and constant number of relevant prior blocks. This methodology prevent the need to carry all information through every layer of a deep transformer and the input-scaling issue in prior ICL works when simulating the training of complex model in-context.*

5 Discussion and Conclusion

We demonstrate a one-layer transformer, with CoT and dynamic masking, simulates multi-step gradient descent in-context in a more efficient way compared to prior work (Section 4). With the CoT nature of recursive generation, we demonstrate its memory saving through the application of ICGD on the statistical model (Section 3). Combined with dynamic masking, the model only needs to process the necessary information from the CoT sequence, and leads to $O(N)$ computational savings compared to prior results on simulating the training of a deep network in-context. In summary, we provide theoretical guarantee on the effectiveness and efficiency of CoT with dynamic masking. This paves the way for enabling transformers to perform complex, multi-step learning algorithms in context with better resource utilization.

Connecting to Practice. The idea of generating very long CoT but tackling the computation issue (quadratic cost of attention) has raised the notice of the community (Giannou et al., 2023; Yang et al., 2025). To produce an infinite-length CoT process to mimic an infinite thinking process, we must find some way to keep only the necessary information to produce the next token. The dynamic masking implements this spirit. Recently, Yang et al. (2025) approach the same intuition as us to solve this problem, but with different method and aspect of theoretical analysis. Specifically, they introduce a reduction rule that deletes token spans once they become irrelevant, and empirically show that a limited-size and context window transformer solves challenging tasks, such as Einstein's puzzle, given a long thinking time. This result backs up the usefulness of our dynamic masking intuition. In contrast to (Yang et al., 2025), our dynamic masking keeps the token in the input sequence, but only lets the attention layer see the necessary tokens. Hence our method doesn't have the risk of deleting information later might be useful, but still eliminates the growth of computation cost along with longer CoT. The theoretical axis we investigate is also different. They prove that a fixed-size and context window transformer generates a token trace that encodes the Turing machine. Complementary to the Turing machine, we show the strength of this architecture from the *in-context learning* perspective with an optimization objective in deep learning and machine learning models. Our theoretical analysis tackles this by proving CoT simulating training deep networks and statistical models, and gives an explicit construction of such transformer and provable approximation error.

Impact Statement

By the theoretical nature of this work, we do not anticipate any negative social impact.

References

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36:57125–57211, 2023.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36, 2024.

Cheng, X., Chen, Y., and Sra, S. Transformers implement functional gradient descent to learn non-linear functions in context. *arXiv preprint arXiv:2312.06528*, 2023.

Cui, Y., He, P., Tang, X., He, Q., Luo, C., Tang, J., and Xing, Y. A theoretical understanding of chain-of-thought: Coherent reasoning and error-aware demonstration. *arXiv preprint arXiv:2410.16540*, 2024.

Dai, D., Sun, Y., Dong, L., Hao, Y., Ma, S., Sui, Z., and Wei, F. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. *arXiv preprint arXiv:2212.10559*, 2022.

Feng, G., Zhang, B., Gu, Y., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36:70757–70798, 2023.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Gatmiry, K., Saunshi, N., Reddi, S. J., Jegelka, S., and Kumar, S. Can looped transformers learn to implement multi-step gradient descent for in-context learning? *arXiv preprint arXiv:2410.08292*, 2024.

Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

Hu, J. Y.-C., Liu, H., Chen, H.-Y., Wu, W., and Liu, H. Universal approximation with softmax attention. *arXiv preprint arXiv:2504.15956*, 2025.

Huang, J., Wang, Z., and Lee, J. D. Transformers learn to implement multi-step gradient descent with chain of thought. *arXiv preprint arXiv:2502.21212*, 2025.

Kim, J. and Suzuki, T. Transformers provably solve parity efficiently with chain of thought. *arXiv preprint arXiv:2410.08633*, 2024.

Li, Y., Sreenivasan, K., Giannou, A., Papailiopoulos, D., and Oymak, S. Dissecting chain-of-thought: Compositionality through in-context filtering and learning. *Advances in Neural Information Processing Systems*, 36: 22021–22046, 2023.

Li, Z., Liu, H., Zhou, D., and Ma, T. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*, 1, 2024.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3. Atlanta, GA, 2013.

Merrill, W. and Sabharwal, A. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023a.

Merrill, W. and Sabharwal, A. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023b.

Pinkus, A. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

Prystawski, B., Li, M., and Goodman, N. Why think step by step? reasoning emerges from the locality of experience. *Advances in Neural Information Processing Systems*, 36: 70926–70947, 2023.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Wang, Z., Jiang, B., and Li, S. In-context learning on function classes unveiled for transformers. In *Forty-first International Conference on Machine Learning*, 2024.

Wu, W., Su, M., Hu, J. Y.-C., Song, Z., and Liu, H. In-context deep learning via transformer models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Yang, C., Srebro, N., McAllester, D., and Li, Z. Pencil: Long thoughts with short memory. *arXiv preprint arXiv:2503.14337*, 2025.

Appendix

Limitations and Future Work

Our guarantees rest on several idealized conditions. First, dynamic masking is treated as cost-free and omniscient. Ee ignore the overhead of computing masks and the latency of repeatedly scanning an ever-growing prompt. Second, the claimed $O(N)$ FLOP saving is asymptotic—the constant factors introduced by serial reuse of the single block and by prompt growth remain unquantified. Lastly, our proofs target ReLU feed-forward networks and convex statistical objectives. Extending the framework to convolutional, residual, or attention-based settings and to adaptive or second-order optimizers is left for future work.

A Supplementary Theoretical Backgrounds

Here we introduce some axillary lemmas and definitions for later convenience.

Lemma A.1 (Proposition A.2 of (Bai et al., 2023), Gradient Descent for Smooth and Strongly Convex Functions)**.** *Let L be α-strongly convex and β-smooth function. Then the gradient descent $w_{\mathrm{GD}}^{t+1} = w_{\mathrm{GD}}^{t} - \eta \nabla L(w_{\mathrm{GD}}^{t})$ with learning rate $\eta = 1/\beta$ satisfies the following two exponential convergence rates*

$$\|w_{\mathrm{GD}}^{t} - w^{\star}\|_2^2 \leq \exp\left(-\frac{t}{\kappa}\right) \cdot \|w_{\mathrm{GD}}^{0} - w^{\star}\|_2^2,$$

$$L(w_{\mathrm{GD}}^{t}) - L(w^{\star}) \leq \frac{\beta}{2} \exp\left(-\frac{t}{\kappa}\right) \cdot \|w_{\mathrm{GD}}^{0} - w^{\star}\|_2^2,$$

where $\kappa := \beta/\alpha$ is the condition number and $w^{\star}$ is the minimizer of L.

Definition A.1 ($B_2^k(R)$)**.** *Let $B_2^k(R)$ denote a ball containing all points in $\mathbb{R}^k$ whose ℓ_2-norm is at most R.*

Lemma A.2 (Lemma 14 of (Bai et al., 2023), Composition of Error for Approximating Convex Gradient Descent)**.** *Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is a convex function. Define optimal solution $w^{\star} := \mathrm{argmin}_{w \in \mathbb{R}^d} f(w)$, $R \geq 2\|w^{\star}\|_2$, and assume that f is β-smooth on $\mathbb{B}_2^d(R)$. We define sequences $\{\widehat{w}^l\}_{l \geq 0} \in \mathbb{R}^d$, with $\{w_{\mathrm{GD}}^l\}_{l \geq 0} \in \mathbb{R}^d$ as follow:*

$$\begin{cases} \widehat{w}^{l+1} = \widehat{w}^{l} - \eta \nabla f(\widehat{w}^{l}) + \epsilon^{l}, & \|\epsilon^{l}\|_2 \leq \varepsilon, \\ w_{\mathrm{GD}}^{l+1} = w_{\mathrm{GD}}^{l} - \eta \nabla f(w_{\mathrm{GD}}^{l}), \end{cases} \quad \text{for all} \quad l \geq 0,$$

where ϵ^l is the l-th iteration approximation error and is bounded by ε. Then as long as learning rate $\eta \leq 2/\beta$, for any iteration $L \leq R/(2\varepsilon)$ we have:

$$\|\widehat{w}^{L} - w_{\mathrm{GD}}^{L}\|_2 \leq L\varepsilon,$$

this means that the approximation error is linear with the number of iterations. We also have:

$$\|\widehat{w}^{L}\|_2 \leq \frac{R}{2} + L\varepsilon \leq R.$$

This ensures the norm of approximation solution $\widehat{w}^L$ remain bounded by R.

Lemma A.3 (Proposition A.3 of (Bai et al., 2023), Proximal gradient Descent for Convex Function)**.** *Let $L = f + h : \mathbb{R}^d \to \mathbb{R}$, where f is convex and β-smooth for some $\beta \geq 0$, and h is a convex function. Then the proximal gradient descent*

$$w_{\mathrm{PGD}}^{t+1} = \mathrm{prox}_{\eta\mathcal{R}}(w_{\mathrm{PGD}}^t - \eta\nabla L(w_{\mathrm{PGD}}^t))$$

with $\eta = 1/\beta$ satisfies the following three for $t \geq 1$:

- *The sequence $\{L(w_{\mathrm{PGD}}^t)\}$ is decreasing.*
- *For any minimizer $w^\star \in \arg\min_{w\in\mathbb{R}^d} L(w)$,*

$$L(w_{\mathrm{GD}}^{t+1}) - L(w^\star) \leq \frac{\beta}{2}\left(\|w_{\mathrm{PGD}}^t - w^\star\|_2^2 - \|w_{\mathrm{PGD}}^{t+1} - w^\star\|_2^2\right),$$

  and since $L(w_{\mathrm{PGD}}^{t+1}) \geq L(w^\star)$ we have

$$\|w_{\mathrm{PGD}}^{t+1} - w^\star\|_2 \leq \|w_{\mathrm{PGD}}^t - w^\star\|_2.$$

- *For $k \geq 1$, $t \geq 0$, we have*

$$L(w_{\mathrm{PGD}}^{t+k}) - L(w^\star) \leq \frac{\beta}{2k}\|w_{\mathrm{PGD}}^t - w^\star\|_2^2.$$

Lemma A.4 (Lemma C.1 of (Bai et al., 2023), Composition of error for approximating convex PGD)**.** *Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is a convex function and $\mathcal{R}$ is a convex regularizer. Define optimal solution $w^\star := \mathrm{argmin}_{w\in\mathbb{R}^d} f(w) + \mathcal{R}(w)$, $R \geq 2\|w^\star\|_2$, and assume that f is β-smooth on $\mathbb{B}_2^d(R)$. We define the sequences transformer approximated weights update as $\{\widehat{w}^l\}_{l\geq 0} \in \mathbb{R}^d$ and the standard gradient descent as $\{w_{\mathrm{PGD}}^l\}_{l\geq 0} \in \mathbb{R}^d$:*

$$\begin{cases} \widehat{w}^{l+1} = \mathrm{prox}_{\eta\mathcal{R}}(\widehat{w}^l - \eta\nabla f(\widehat{w}^l)) + \epsilon^l, & \|\epsilon^l\|_2 \leq \varepsilon, \\ w_{\mathrm{GD}}^{l+1} = \mathrm{prox}_{\eta\mathcal{R}}(w_{\mathrm{GD}}^l) - \eta\nabla f(w_{\mathrm{GD}}^l), \end{cases} \quad \textit{for all} \quad l \geq 0,$$

where ϵ^l is the l-th iteration approximation error and is bounded by ε. Then as long as learning rate $\eta \leq 2/\beta$, for any iteration $L \leq R/(2\varepsilon)$ we have

$$\|\widehat{w}^L - w_{\mathrm{GD}}^L\|_2 \leq L\varepsilon.$$

This means the approximation error is linear with the number of iterations. We also have

$$\|\widehat{w}^L\|_2 \leq \frac{R}{2} + L\varepsilon \leq R.$$

This ensures the norm of approximation solution $\widehat{w}^L$ remain bounded by R.

Lemma A.5 (Theorem 3.1 of Hu et al. (2025), Attention Approximates Truncated Linear Models In-Context)**.** *Fix real numbers $a < b$. Let the input be*

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ w & w & \cdots & w \\ t_1 & t_2 & \cdots & t_n \end{bmatrix} \in \mathbb{R}^{(2d+1)\times n},$$

where $\{w, x_i\}_{i\in[n]}$ are bounded. For a precision parameter $p > n$, there exists a single-layer, single-head self-attention Attn *with a linear transformation $A : \mathbb{R}^{(2d+1)\times n} \to \mathbb{R}^{(2d+d_o+2)\times p}$, such that* Attn $\circ A : \mathbb{R}^{d\times n} \to \mathbb{R}^{d_o\times n}$ *satisfies, for any $i \in [n]$,*

$$\begin{aligned} &\|\mathrm{Attn} \circ A(X)_{:,i} - \mathrm{Range}_{[a,b]}(w_i^\top x_i + t_i)e_{\widetilde{k}_i}\|_\infty \\ &\leq \underbrace{\max\{|a|,|b|\}\cdot\epsilon_0}_{\textit{finite-}\beta\textit{ softmax error}} + \underbrace{\frac{b-a}{p}}_{\textit{interpolation error}}. \end{aligned}$$

Here $e_{\widetilde{k}_i}$ is a one-hot vector with a value of 1 at the $\widetilde{k}_i$-th index and 0 elsewhere, $\epsilon_0 = O(e^{-\beta\delta})$, and $\widetilde{k}_i \in [d_o]$ is

$$k_i = \underset{k\in\{0,1,\cdots,p-1\}}{\operatorname{argmin}} (-2x_i^\top w_i - 2t_i + \widetilde{L}_0 + \widetilde{L}_k) \cdot k,$$
$$\widetilde{k}_i = G(k_i).$$

Here $G : [p] \to [d_o]$ denotes any set-to-set function sending each selected interpolation index k_i into an appropriate integer $\widetilde{k}_i \in [d_o]$ for $i \in [n]$.

Remark A.1. *Lemma A.5 prove attention mechanism approximation ReLU activation in-context. Hence by multi-head attention, we add up each head's output to approximate the computation of N-layer ReLU neural network.*

B Related Work

Theoretical Analysis of Chain-of-Thought. Feng et al. (2023) use circuit complexity theory to prove that when solving basic arithmetic problems, a depth-limited transformer need super-polynomially growing model size with respect to input length, while an autoregressive Transformer of constant size solves the tasks by generating CoT in a formal math language. Li et al. (2023) prove transformer in-context learns a L-layer FFN's output with arbitrary error when incorporating the intermediate output of multiple layer FFN into the prompt, which they call CoT prompt. Specifically, they formulate multilayer FFN as a compositional function $f := f_L \circ \cdots f_1$. With input x_i, denote intermediate output of each layer as $s_i^l = f_l(s_i^{l-1})$, the CoT prompt with n sample is in the form $(x_1, s_1^1, \cdots, s_1^L; \cdots, x_n, s_n^1, \cdots; s_n^L)$. By leveraging a filtering mechanism to isolate the tokens relevant to each layer, the transformer decouples the task into simpler linear regression subproblems for each f_l. Kim & Suzuki (2024) prove when training a one-layer transformer with ground-truth intermediate steps, it solve the hard k-parity problem in single gradient update. Huang et al. (2025) show that the global minimizer of one-layer linear transformer trained on CoT-style loss for linear regression performs multi-step gradient descent. Unlike this work, we focus on a problem different from the training dynamic when equipped transformer with CoT objectives: a theoretical proof of a *softmax* transformer with CoT input simulates multi-step gradient descent on various statistical model losses, and how dynamic-masking further improves the efficiency of inference time CoT. Prystawski et al. (2023) study CoT from a probabilistic perspective. They formulate the reasoning process of CoT as a sequential process of estimating conditional probabilities of intermediate variables. When the training data reveals local structure, they show CoT performs better than direct inference. Cui et al. (2024) study CoT by comparing it to In-context learning. Their results show that under assumptions on model parameters, CoT achieves lower error compared to ICL. Further, they also show that CoT is more sensitive to noise being introduced to intermediate steps compared to ICL. Li et al. (2024); Merrill & Sabharwal (2023a) both study CoT from the perspective of circuit complexity. Merrill & Sabharwal (2023b) show that the expressive power of constant-depth transformers is bounded, and CoT provides higher expressiveness by breaking the limitation of constant depth.

In-Context Gradient Descent of Neural Network. Despite (Wu et al., 2025), several other works investigate the transformer's ability to simulate gradient descent training of neural networks through in-context. Early studies focused on simple models. For example, Akyürek et al. (2022) demonstrate a transformer is capable of performing gradient updates for linear regression in an implicit fashion. Similarly, Dai et al. (2022) argue GPT-style models function as meta-optimizers performing gradient descent. Subsequent works further extend these insights. Von Oswald et al. (2023) provide a constructive proof that a transformer is capable of carrying out multiple steps of gradient descent internally. Bai et al. (2024) further show that transformers can even select among different learning algorithms in context (hence earning the moniker "transformers as statisticians"), provably adapting to the best solver for a given task. Importantly, Bai et al. (2024) also show that a frozen transformer imitates gradient descent for two-layer networks, but their technique does not extend to deeper models because it avoids explicit backpropagation. In contrast, we prove this in a CoT manner that the one-layer transformer simulates L GD steps on N-layer network. This is the first formal evidence that the "thinking steps" written in a CoT prompt implement back-propagation. By establishing this result, we clarify how LLMs might learn complex learning algorithms by CoT prompting. Giannou et al. (2023) treat transformer as a looped computer and proves by recursively using 13 transformer layers, it simulates training of N-layer sigmoid neural network. In contrast, our paper uses only one-layer transformer block to approximate training of N-layer ReLU neural network, which is also more practical in real-world use of neural network activation (Glorot et al., 2011; Maas et al., 2013; Ramachandran et al., 2017). Also, the input sequence length for the looped transformer in Giannou et al. (2023) increase with the number of network layers N. This is because executing more sequential operations (more forward and backward steps for deeper networks during gradient descent) requires an increased number of "instruction" in the input sequence by their design. Gatmiry et al. (2024) further study the training dynamic of linear looped transformer, showing that the global minimizer on the empirical risk implements multi-step preconditioned gradient descent on the linear regression instance. In contrast, we study constructive expressivity of a practical softmax Transformer. We show that a fixed Transformer, run autoregressively with CoT traces, implements multi-step ICGD for multiple statistical objectives (GLM, ridge, lasso) and simulates full forward–backpropagation for N-layer neural network.

CoT with Dynamic Masking: Efficiency over (Wu et al., 2025). Wu et al. (2025) show transformers can simulate the training of an N-layer neural network in-context. Their method needs a gigantic prompt, encoding all weights and activations at every layer. This causes redundant computation because all information moves through each layer. Input length also grows with the simulated model size. This limits the complexity of models learned in-context with fixed input size. We address these issues by using chain-of-thought (CoT) prompting with dynamic masking. Instead of stacking many transformer layers, we reuse one transformer block across L gradient descent steps. At each step, we mask the transformer's

attention. Each attention head only sees tokens needed for the current update. This prevents unnecessary information from moving forward. Our method reduces computation by $O(N)$ compared to (Wu et al., 2025). It simulates training an N-layer network with constant transformer depth (i.e., $O(1)$ depth in a loop and reused repeatedly), not depth $O(N)$. This also removes the fixed-input-size limitation. The prompt can grow with CoT, enabling more complex models and longer training.

C Proofs of Main Text

C.1 Proof of Theorem 3.1

Consider data $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Define the empirical loss

$$\mathcal{L}_{\mathrm{GLM}}(w) = \sum_{i=1}^{n} \ell_{\mathrm{GLM}}(w^\top x_i, y_i), \quad \text{with} \quad \ell_{\mathrm{GLM}}(s, y) = -ys + \int_0^s g(t)\, dt,$$

where the link function $g : \mathbb{R} \to \mathbb{R}$ is continuous.

Define the input as:

$$X := \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ w_{\mathrm{CoT}}^{(0)} & w_{\mathrm{CoT}}^{(0)} & \cdots & w_{\mathrm{CoT}}^{(0)} \\ y_1 & y_2 & \cdots & y_n \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

where $w_{\mathrm{CoT}}^{(0)} = 0_d \in \mathbb{R}^d$ is the initialization of the generalized linear model. We set $Z_1 = X$.

Theorem C.1 (Restated of Theorem 3.1, Chain-of-Thought Gradient Descent Approximation of GLM.)**.** *Assume that L_{GLM} is α-strongly convex and β-smooth on the ball $\mathbb{B}_2(B_w)$, where B_w satisfies $\|w_{\mathrm{GLM}}\|_2 \le B_w/2$, and $w_{\mathrm{GLM}} = \arg\min_w \mathcal{L}_{\mathrm{GLM}}(w)$. Under this setting, there exists a multi-head attention mechanism Attn_m with a linear mapping A and a feedforward network FFN_1 such that at each step l, the transformer approximates a single gradient update:*

$$\|\mathrm{Attn}_{\mathrm{m}} \circ A(Z_l) - \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ w^{(l)} - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) & \cdots & w^{(l)} - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) \\ 1 & \cdots & 1 \end{bmatrix} \|_\infty \le \epsilon.$$

For $\eta \le 2/\beta$ and $L \le B_w/(2\epsilon)$, after L iterations, the final CoT weight $w_{\mathrm{CoT}}^{(L)}$ approximates the true minimizer w_{GLM} with the error bound

$$\|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{GLM}}\|_2 \le L\epsilon + e^{-(L/2\kappa)} \frac{B_w}{2}.$$

Proof. The derivative of the GLM loss with respect to s is $\partial_s \ell(s, y) = -y + g(s)$. Since by assumption g is continuous, $\partial_s \ell$ is also continuous.

Therefore by Lemma 3.1, there exists a multi-head attention $\mathrm{Attn}_{\mathrm{m}}$ with skip connections and a linear mapping A such that

$$\left\| \mathrm{Attn}_{\mathrm{m}} \circ A(X) - \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ w - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) & \cdots & w - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) \\ 1 & \cdots & 1 \end{bmatrix} \right\|_\infty \le \epsilon. \tag{5}$$

We begin to show how to incorporate the above result into CoT setting.

At iteration l, assume the transformer's current state (CoT prompt) is given by

$$Z_l = \begin{bmatrix} x_1 & \cdots & x_n & x_1 & \cdots & x_n & \cdots & x_1 & \cdots & x_n \\ w^{(1)} & \cdots & w^{(1)} & w^{(2)} & \cdots & w^{(2)} & \cdots & w^{(l)} & \cdots & w^{(l)} \\ y_1 & \cdots & y_n & y_1 & \cdots & y_n & \cdots & y_1 & \cdots & y_n \\ 1 & \cdots & 1 & 1 & \cdots & 1 & \cdots & 1 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{(2d+3)\times ln}.$$

In this prompt, each block of n columns corresponds to a previous gradient descent update, with $w^{(l)}$ denoting the weight at the l-th step.

Suppose we set the context windows of the model to be n, the input of become G_l as defined in Definition 2.1

$$G_l = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ w^{(l)} & w^{(l)} & \cdots & w^{(l)} \\ y_1 & y_2 & \cdots & y_n \\ 1 & 1 & \cdots & 1 \end{bmatrix}.$$

Combine with (5), we have

$$\|\mathrm{Attn}_{\mathrm{m}} \cdot A \cdot (G_l) - \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ w^{(l)} - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) & \cdots & w^{(l)} - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) \\ 1 & \cdots & 1 \end{bmatrix} \|_\infty \leq \epsilon. \tag{6}$$

Next we compute the approximation error of performing L-th step approximation. We apply Lemma A.2 to achieve this goal.

First we need to change the error bound in terms of ℓ_2 norm.

We write each iteration of the gradient update in the CoT prompt as

$$w_{\mathrm{CoT}}^{(l+1)} = w^{(l)} - \eta \nabla_w \mathcal{L}_{\mathrm{GLM}}(w) + \varepsilon^{(l)},$$

and by (6) an ℓ_∞ error at most ϵ, we bound the ℓ_2-error as

$$\|\varepsilon^{(l)}\|_2 \leq \sqrt{2d+3} \cdot \|\varepsilon^{(l)}\|_\infty \leq \sqrt{2d+3} \cdot \epsilon,$$

where we later absorb the $\sqrt{2d+2}$ into ϵ.

Now by precondition we have $\eta \leq \beta/2$ and $L \leq B_w/(2\epsilon)$, and by setting $w^{(0)} = w_{\mathrm{CoT}}^{(0)} = 0_d$ we have

$$\|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{GD}}^{(L)}\|_2 \leq L\epsilon \quad \text{and} \|w_{\mathrm{CoT}}^{(L)}\| \leq B_w. \qquad \text{(By Lemma A.2)}$$

Since $w_{\mathrm{GD}}^{(L)}$ converges to the true minimizer w_{GLM} as long as the empirical loss L_{GLM} is α-strongly convex and β-smooth (Lemma A.1), we conclude that the CoT-based iterates $w_{\mathrm{CoT}}^{(L)}$ also converge to w_{GLM} in the following way

$$\begin{aligned} \|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{GLM}}\|_2 &\leq \|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{GD}}^{(L)}\|_2 + \|w_{\mathrm{GD}}^{(L)} - w_{\mathrm{GLM}}\|_2 \\ &\leq L\epsilon + e^{-(L/2\kappa)} \|w_{\mathrm{GLM}}\|_2 \\ &\leq L\epsilon + e^{-(L/2\kappa)} \frac{B_w}{2}. \qquad \text{(By Lemma A.1)} \end{aligned}$$

This completes our proof. □

C.2 Proof of Theorem 3.2

We restate the ridge regression as follows. Consider the ridge regression estimators $w_{\text{ridge}}^{\lambda}$ over the in-context examples in $\mathcal{D}$ with regularization $\lambda \geq 0$:

$$w_{\text{ridge}}^{\lambda} \coloneqq \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \mathcal{L}_{\text{ridge}}(W) = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (\langle w, x_i \rangle - y_i)^2 + \frac{\lambda}{2} \|w\|_2^2 \right\}.$$

We state the CoT ICGD on ridge regression empirical risk by single attention layer.

Theorem C.2 (Restated of Theorem 3.2, Chain-of-Thought ICGD Approximation of Ridge Regression.)**.** *Fix $\lambda \geq 0$ and assume $\|w_{\text{ridge}}^{\lambda}\|_2 \leq B_w/2$. Let $\eta \leq 2/\beta$ and choose the CoT length $L \leq \frac{B_w}{2}/(2\epsilon)$. Then there exists a multi-head attention layer with a linear map such that the CoT iterates satisfy*

$$\|w_{\text{CoT}}^{(L)} - w_{\text{ridge}}^{\lambda}\|_2 \leq L\epsilon + e^{-L/(2\kappa)} \frac{B_w}{2}.$$

Proof. The Hessian of ridge regression is

$$H = \frac{1}{N} X X^\top + \lambda I_d.$$

The ridge regression loss is known to be α-strongly convex and and β-smooth with

$$\alpha = \lambda_{\min}(H), \quad \beta = \lambda_{\max}(H), \quad , \kappa = \beta/\alpha.$$

Since $\partial \ell(s, y)/\partial_s = s - y$ is continuous for least square loss, and ℓ_2 penalty contribute λw to the gradient is also C^1. Hence by Lemma 3.1, for every $\epsilon \geq 0$ there exists a single-layer multi-head attention with a linear projection denoted as $\text{Attn}_{\text{m}} \cdot \text{A}$ such that

$$\left\| \text{Attn}_m \circ A(G_l) - \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \\ w_{\text{CoT}}^{(l)} - \eta \nabla_w \mathcal{L}_{\text{ridge}}(w_{\text{CoT}}^{(l)}) & \dots & w_{\text{CoT}}^{(l)} - \eta \nabla_w \mathcal{L}_{\text{ridge}}(w_{\text{CoT}}^{(l)}) \\ l+1 & \dots & l+1 \\ 1 & \dots & 1 \end{bmatrix} \right\|_\infty \leq \epsilon, \tag{7}$$

with learning-rate $\eta \leq 2/\beta$.

Next we bound the error composition along CoT by Lemma A.2. We write the single-step CoT gradient descent approximation as

$$w_{\text{CoT}}^{(l+1)} = w_{\text{CoT}}^{(l)} - \eta \nabla_w \mathcal{L}_{\text{ridge}}(w_{\text{CoT}}^{(l)}) + \varepsilon^{(l)},$$

where $\|\varepsilon^{(l)}\|_\infty \leq \epsilon$ according to (7). We convert the ℓ_∞ error into ℓ_2 and absorb the $\sqrt{2d+3}$ into ϵ to have $\|\varepsilon^{(l)}\|_2 \leq \epsilon$.

By the Lemma A.2 and choose

$$L \leq B_w/(2\epsilon), \quad B_w \geq 2\|w_{\text{ridge}}^{\lambda}\|_2,$$

we have

$$\|w_{\text{CoT}}^{(L)} - w_{\text{GD}}^{(L)}\|_2 \leq L\epsilon, \quad \|w_{\text{CoT}}^{(L)}\|_2 \leq B_w. \tag{8}$$

Because $\mathcal{L}_{\text{ridge}}$ is α-strongly convex and β-smooth, Lemma A.1 gives exponential convergence of standard gradient descent

$$\|w_{\text{GD}}^{(L)} - w_{\text{ridge}}^{\lambda}\|_2 \leq e^{-L/(2\kappa)} \|w_{\text{ridge}}^{\lambda}\|_2. \tag{9}$$

Combining (8) and (9) we have

$$\|w_{\mathrm{CoT}}^{(L)} - w_{\mathrm{ridge}}^{\lambda}\|_2 \leq L\epsilon + e^{-L/(2\kappa)}\|w_{\mathrm{ridge}}^{\lambda}\|_2.$$

This completes the proof. □

C.3 Proof of Theorem 3.3

We consider approximating proximal gradient descent for lasso regression. Specifically, the lasso loss is defined as follows

$$\mathcal{L}_{\text{lasso}} = \frac{1}{2n}\sum_{i=1}^{n}(w^\top x_i - y_i)^2 + \lambda\|W\|_1.$$

The classical method for minimizing losses with a non-smooth regularizer (such as ℓ_1 norm) is Proximal Gradient Descent (PGD). The single-step PGD updates parameters via stardard gradient steps followed by a proximal operator $\text{prox}_{\eta\mathcal{R}}$

$$w_{\text{PGD}}^{t+1} \coloneqq \text{prox}_{\eta\mathcal{R}}\left(w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t)\right), \quad \text{where} \quad \mathcal{L}_n^0(w) \coloneqq 1/n\sum_{i=1}^{n}(w^\top x_i - y_i)^2. \tag{10}$$

Theorem C.3 (Restated of Theorem 3.3, Chain-of-Thought Proximal ICGD Approximation of Lasso Regression)**.** *Given $\mathcal{L}_n^0(w) \coloneqq 1/N\sum_{i=1}^{N}(w^\top x_i - y_i)^2$ and the proximal operator $\text{prox}_{\eta\lambda\|\cdot\|_1}$ associated with the Lasso regularizer, there exists a Transformer layer consisting of a multi-head attention module Attn_m, linear transform A, and one FFN layers FFN, that approximates a single proximal gradient descent (PGD) step in (10) for Lasso regression to arbitrary precision. Specifically, for any $\epsilon \geq 0$, we have*

$$\|\text{FFN}_2 \circ \text{Attn}_\text{m} \circ A(Z_l) - \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ \text{prox}_{\eta\lambda\|\cdot\|_1}(w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t)) & \cdots & \text{prox}_{\eta\lambda\|\cdot\|_1}(w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t)) \\ 1 & \cdots & 1 \end{bmatrix}\|_\infty \leq \epsilon.$$

For $\eta \leq 2/\beta$ and $L \leq R/(2\epsilon)$, after L iterations, the final CoT weight $w_{\text{CoT}}^{(L)}$ approximates the true minimizer w_{GLM}^* in terms of loss function

$$\mathcal{L}_{\text{lasso}}(w_{\text{CoT}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{lasso}}^*) \leq (C_0 + \beta R + \lambda\sqrt{d})L\epsilon + \frac{\beta}{2}(L\epsilon)^2 + \frac{\beta R^2}{8L}.$$

Proof Sketch. The proof consist of two steps. First we approximate the standard gradient descent step of least square loss, and second the the proximal operator. We approximate the gradient descent step by multihead attention according to Lemma 3.1. For the proximal operator, we rely on (Bai et al., 2023)[Proposition C.1] that the proximal operator for the ℓ_1-norm regularizer is exactly approximable by FFN layer. □

Proof. The proof is similar to the proof in Theorem 3.1. First, since $\mathcal{L}_n^0(w)$ is a C^1 function, by **??** we know there exist a multihead attention Attn_m with skip connection, combined with linear transform A, and a selection network FFN_1, such that for input Z_l defined in Definition 2.1 we have

$$\|\text{Attn}_\text{m} \circ A \circ \text{FFN}_1(Z_l) - \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t) & \cdots & w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t) \\ 1 & \cdots & 1 \end{bmatrix}\|_\infty \leq \epsilon.$$

Thus we approximate the gradient-descent step (without the proximal operator) via an attention layer.

Next, we approximate the proximal operator $\text{prox}_{\eta\lambda\|\cdot\|_1} : \mathbb{R}^d \to \mathbb{R}^d$. By (Bai et al., 2023, Definition C.1, Proposition C.1), we know there exist a $\text{FFN}^{\text{prox}} : \mathbb{R}^d \to \mathbb{R}^d$ with weight matrices

$$W_1^{\text{prox}} \in \mathbb{R}^{4d\times d}, W_2^{\text{prox}} \in \mathbb{R}^{d\times 4d},$$

such that it approximate $\text{prox}_{\eta\lambda\|\cdot\|_1}(\cdot)$ exactly

$$\text{FFN}^{\text{prox}}(w) = W_2^{\text{prox}}\text{ReLU}(W_1^{\text{prox}}(w)) = \text{prox}_{\eta\lambda\|\cdot\|_1}(w). \tag{11}$$

However, the current token dimension is $D = 2d + 3$, and the weight vector we wish to update is in the middle of each token. To handle this, we embed the FFN^{prox} into a higher-dimensional FFN_2, while leaving all other coordinates unchanged.

Specifically, let P be a permutation matrix $\mathbb{P} \in \mathbb{R}^{D\times D}$ that moves the w coordinates to the front and keeps the rest order preserved

$$Pz_i = P[x_i; y_i; w; l; 1]^\top = [w; x_i; y_i; l; 1]^\top := [w; \widetilde{z}_i]^\top.$$

Then we construct the new weights $(W_{1'}^{\text{prox}}, W_{2'}^{\text{prox}})$ as follows

$$W_{1'}^{\text{prox}} = \begin{bmatrix} W_1^{\text{prox}} & 0 \\ 0 & I_{d+3} \\ 0 & -I_{d+3} \end{bmatrix}, \quad W_{2'}^{\text{prox}} = \begin{bmatrix} W_2^{\text{prox}} & 0 & 0 \\ 0 & I_{d+3} & -I_{d+3} \end{bmatrix}. \tag{12}$$

The output of the first linear layer $W_{1'}^{\text{prox}}$ after ReLU activation is

$$\text{ReLU}(W_{1'}^{\text{prox}} P z_i) = [\text{ReLU}(W_1^{\text{prox}} w); \text{ReLU}(\widetilde{z}_i); \text{ReLU}(-\widetilde{z}_i)]^\top.$$

Pass it through the second linear layer $W_{2'}^{\text{prox}}$ we derive

$$\begin{aligned} & W_{2'}^{\text{prox}} \text{ReLU}(W_{1'}^{\text{prox}} P z_i) \\ &= [W_2^{\text{prox}} \text{ReLU}(W_1^{\text{prox}} w); \text{ReLU}(\widetilde{z}_i) - \text{ReLU}(-\widetilde{z}_i)]^\top && \text{(By (12))} \\ &= [\text{prox}_{\eta\lambda\|\cdot\|_1}(w); \widetilde{z}_i], \end{aligned}$$

where the last inequality comes from (11) and the ReLU trick that $\text{ReLU}(z) - \text{ReLU}(-z) = z$.

Applying $P^\top$ to reorder the coordinates back yields the desired result

$$P^\top W_{2'}^{\text{prox}} \text{ReLU}(W_{1'}^{\text{prox}} P z_i) = [x_i; y_i; \text{prox}_{\eta\lambda\|\cdot\|_1}(w); l; 1]^\top.$$

Combining both steps, the overall transformation by attention and plus one FFN with weight matrices $(W_{1'}^{\text{prox}} P, P^\top W_{2'}^{\text{prox}})$ approximates single PGD step for the lasso loss as follow

$$\|\text{FFN}_2 \circ \text{Attn}_{\text{m}} \circ A(Z_l) \tag{13}$$

$$- \begin{bmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \\ \text{prox}_{\eta\lambda\|\cdot\|_1}(w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t)) & \cdots & \text{prox}_{\eta\lambda\|\cdot\|_1}(w_{\text{PGD}}^t - \eta\nabla\mathcal{L}_n^0(w_{\text{PGD}}^t)) \\ 1 & \cdots & 1 \end{bmatrix} \|_\infty \le \epsilon. \tag{14}$$

Until now, we finish the proof of single-PGD step approximation. We now move to error accumulation for multi-step approximation.

Denote the transformer approximated result by $w_{\text{CoT}}^{(l)}$ and the exact PGD result by $w_{\text{PGD}}^{(l)}$. Each step introduces an error bounded by ϵ by (13), we write it in the form of Lemma A.4

$$w_{\text{CoT}}^{(l+1)} = \text{prox}_{\eta\lambda\|\cdot\|_1}(w_{\text{CoT}}^{(l)} - \eta\nabla\mathcal{L}_n^0(w_{\text{CoT}}^{(l)})) + \varepsilon^{(l)}, \quad \text{with} \quad \|\varepsilon^{(l)}\|_\infty \le \epsilon.$$

We convert the ℓ_∞ error into ℓ_2 and absorb the $\sqrt{2d+3}$ into ϵ to have $\|\varepsilon^{(l)}\|_2 \le \epsilon$. By setting $w^{(0)} = w_{\text{CoT}}^{(0)} = 0_d$ and keeping $\eta \le 2/\beta$, Lemma A.4 yields for any $L \le R/(2\epsilon)$ we have

$$\|w_{\text{CoT}}^{(L)} - w_{\text{PGD}}^{(L)}\| \le L\epsilon, \quad \|\|w_{\text{CoT}}^{(L)}\|_2 \le R.$$

Furthermore, Lemma A.3 guarantees

$$\mathcal{L}_{\text{lasso}}(w_{\text{PGD}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{lasso}}^*) \le \frac{\beta}{2L}\|w_{\text{lasso}}^*\|_2^2 \le \frac{\beta R^2}{8L}. \tag{15}$$

Now we can calculate $\mathcal{L}_{\text{lasso}}(w_{\text{CoT}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{lasso}}^*)$.

We decompose the error as

$$\begin{aligned}\mathcal{L}_{\text{lasso}}(w_{\text{CoT}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{lasso}}^{*}) =&(\mathcal{L}_{\text{lasso}}(w_{\text{CoT}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{PGD}}^{(L)})) \\ &+ (\mathcal{L}_{\text{lasso}}(w_{\text{PGD}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{lasso}}^{*})).\end{aligned}$$

The second term is bounded by (15), hence we focus on the first term.

Define $\Delta = w_{\text{CoT}}^{(L)} - w_{\text{PGD}}^{(L)}$, thus $\|\Delta\|_2 \le L\epsilon$. Since $\mathcal{L}_n^0(w)$ is β-smooth, apply the descent lemma we have

$$\begin{aligned}\widehat{L}_N^0(w_{\text{CoT}}^{(L)}) - \widehat{L}_N^0(w_{\text{PGD}}^{(L)}) &\le \nabla\widehat{L}_N^0(w_{\text{PGD}}^{(L)})^\top\Delta + \frac{\beta}{2}\|\Delta\|_2^2 \\ &\le (C_0 + \beta R)\|\Delta\|_2 + \frac{\beta}{2}\|\Delta\|_2^2 \\ &\le (C_0 + \beta R)L\epsilon + \frac{\beta}{2}(L\epsilon)^2.\end{aligned}$$

where we bound the gradient norm $\|\nabla\widehat{L}_N^0(w_{\text{PGD}}^{(L)})\|$ by applying β-smoothness of $\widehat{L}_N^0$ again, but in the view that its gradient is β-lipschitz, that is for all u and v

$$\|\nabla\widehat{L}_N^0(u) - \nabla\widehat{L}_N^0(v)\| \le \beta\|u - v\|.$$

By setting $v = w_{\text{PGD}}^{(0)} = 0$ and $u = w_{\text{PGD}}^{(L)}$ with $\|u\|_2 \le R$ we have

$$\begin{aligned}\|\nabla\widehat{L}_N^0(w_{\text{PGD}}^{(L)})\|2 &= \|\nabla\widehat{L}_N^0(w_{\text{PGD}}^{(L)}) - \nabla\widehat{L}_N^0(0)\|_2 + \|\nabla\widehat{L}_N^0(0)\|_2 \\ &\le \beta R + C_0,\end{aligned}$$

where $C_0 := |\nabla\widehat{L}_N^0(0)|_2$.

For the regularizer, we use the lipschitz continuity of the ℓ_1 norm,

$$|\|w_{\text{CoT}}^{(L)}\|_1 - \|w_{\text{PGD}}^{(L)}\|_1| \le \sqrt{d}\|\Delta\|_2.$$

Hence we have

$$\mathcal{L}_{\text{lasso}}(w_{\text{CoT}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{PGD}}^{(L)}) \le (C_0 + \beta R + \lambda\sqrt{d})L\epsilon + \frac{\beta}{2}(L\epsilon)^2.$$

Combining the above two parts, we get the final bound

$$\mathcal{L}_{\text{lasso}}(w_{\text{CoT}}^{(L)}) - \mathcal{L}_{\text{lasso}}(w_{\text{lasso}}^{*}) \le (C_0 + \beta R + \lambda\sqrt{d})L\epsilon + \frac{\beta}{2}(L\epsilon)^2 + \frac{\beta R^2}{8L}. \tag{16}$$

We discuss in which case the standard proximal gradient descent convergence rate dominate (the last term). Let $C := C_0 + \beta R + \lambda\sqrt{d}$.

According to the proof of Lemma 3.1 original paper from Hu et al. (2025), the approximation error is consist of $\epsilon = dH\epsilon_1 + d\,d\epsilon_0$. By their proof and classical result of ReLU neural network approximation from (Pinkus, 1999, Theorem 3.1) we have

$$\epsilon = O(d(\frac{H}{p} + H^{-\alpha})),$$

where $\alpha = k/d$, k means the target function is k-times continuously differentiable, in the least square case $\alpha = 1/2$.

Balance the two contributions in ϵ by setting $p = H^{3/2}$, then

$$\epsilon = O(dH^{-1/2}).$$

Solving Equation (16), the last term dominates when

$$\epsilon \leq \frac{R^2}{8CL^2}.$$

Insert to above equation give

$$d\,H^{-1/2} \leq \frac{R^2}{8C\,L^2}, \quad H \geq \Big(\frac{8C\,d}{R^2}\Big)^2 L^4.$$

In conclusion, choosing

$$H = O(L^4)$$

make $\epsilon \leq R^2/(8CL^2)$ so that

$$(C_0 + \beta R + \lambda\sqrt{d})L\epsilon + \frac{\beta}{2}(L\epsilon)^2 = O(1/L)$$

This completes the proof. □

C.4 Proof of Theorem 4.1

Theorem C.4 (In-context Calculation of Forward Propagation)**.** *Let l denote the current round. When $t \bmod 2N \in [N]$, the network performs forward propagation for the N-layer feed-forward network. The input it receives from the dynamic masking mechanism is*

$$\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix} = \begin{bmatrix} o_{i-1} \cdot 1_{1\times d} \\ o_{i-2} \cdot 1_{1\times d} \\ \sigma'(z_{i-1}) \cdot 1_{1\times d} \\ \mathrm{vec}(W_i) \cdot 1_{1\times d} \\ 1_{1\times d} \\ \delta_i \cdot 1_{1\times d} \\ (W_i^{l-1})^\top \delta_i \cdot 1_{1\times d} \\ \underbrace{(\mathrm{vec}(W_i^{l-1}) - \eta \mathrm{vec}(\delta_i o_i^\top))}_{\mathrm{vec}(W_i^l)} \cdot 1_{1\times d} \\ \mathrm{vec}(W_{i-1}) \cdot 1_{1\times d} \\ 0_{(d+1)\times d} \end{bmatrix}.$$

For any $\epsilon > 0$, there exists a single-layer transformer network $\mathcal{F}_{\mathrm{forward}}$ with positional encoding such that

$$\left\| \mathcal{F}_{\mathrm{forward}}\left(\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix} \right) - F_i \right\|_\infty \le \epsilon,$$

where

$$F_i = \begin{bmatrix} o_i \cdot 1_{1\times d} \\ o_{i-1} \cdot 1_{1\times d} \\ \sigma'(z_i) \cdot 1_{1\times d} \\ \mathrm{vec}(W_i) \cdot 1_{1\times d} \\ 1_{1\times d} \end{bmatrix}.$$

The guarantee holds except for an arbitrarily small region in the input space.

Proof. Our goal is to explicitly construct a *single layer transformer block* that maps the input $\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix}$ to (an approximation of) F_i. Essentially, the construction is organized to mirror the forward update we aim to simulate. We will do this in the following manner:

1. extract from the input the two quantities needed to form the pre-activation z_i;
2. use d attention heads to compute (approximately) the broadcasted pre-activation $z_i \cdot \mathbf{1}_{1\times d}$;
3. use a token wise FFN to produce the two nonlinear rows $\sigma(z_i) \cdot \mathbf{1}_{1\times d}$ and $\sigma'(z_i) \cdot \mathbf{1}_{1\times d}$ from this approximation;
4. route the remaining (linear) rows to the output via skip connections, and then bound the total ℓ_∞ error.

Step 1: Extract the needed rows by a token wise linear map. We first apply a token wise linear transformation that extracts the rows containing o_{i-1} and $\mathrm{vec}(W_i)$.

$$A\left(\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix} \right) = \begin{bmatrix} o_{i-1} \cdot \mathbf{1}_{1\times d} \\ \mathrm{vec}(W_i) \cdot \mathbf{1}_{1\times d} \end{bmatrix} \in \mathbb{R}^{(d+d^2)\times d}. \tag{17}$$

One explicit choice is the block matrix

$$A = \begin{bmatrix} I_d & 0 & 0 & 0 \\ 0 & 0 & I_{d^2} & 0 \end{bmatrix},$$

where 0 denotes zero blocks of the appropriate dimensions.

Step 2: Use attention to compute $z_i \cdot \mathbf{1}_{1\times d}$. Our next goal is to produce the pre activation $z_i \cdot \mathbf{1}_{1\times d}$, since then the transformer FFN can apply $\sigma(\cdot)$ entrywise to obtain $o_i \cdot \mathbf{1}_{1\times d} = \sigma(z_i) \cdot \mathbf{1}_{1\times d}$.

Fix $h \in [d]$. To apply Lemma A.5 we need a length-d vector derived from W_i. Let $S_h \in \mathbb{R}^{d\times d^2}$ be a fixed selection matrix such that

$$S_h \operatorname{vec}(W_i) = w_{i,h} \in \mathbb{R}^d, \tag{18}$$

where $w_{i,h}$ satisfies

$$(z_i)_h = w_{i,h}^\top o_{i-1}. \tag{19}$$

(Equivalently, $w_{i,h}$ is the appropriate row/column slice of W_i consistent with the definition of z_i.) Since S_h is linear, we may absorb it into the head's linear $Q/K/V$ projections by composition; thus the resulting module is still a valid attention head.

By Lemma A.5, for any $\epsilon_1 > 0$ there exists an attention head $\mathrm{Attn}_h^{(1)}$ (with positional encodings) such that

$$\left\| \mathrm{Attn}_h^{(1)}\left(\begin{bmatrix} o_{i-1}\cdot \mathbf{1}_{1\times d} \\ \operatorname{vec}(W_i)\cdot \mathbf{1}_{1\times d} \end{bmatrix} \right) - (z_i)_h\, e_h^{(d)} \cdot \mathbf{1}_{1\times d} \right\|_\infty \le \epsilon_1.$$

That is, head h approximates the h-th coordinate block $(z_i)_h\, e_h^{(d)}$ (across all tokens). Summing these d heads reconstructs the full vector z_i (across all tokens):

$$\begin{aligned}
&\left\| \sum_{h=1}^d \mathrm{Attn}_h^{(1)}\left(\begin{bmatrix} o_{i-1}\cdot \mathbf{1}_{1\times d} \\ \operatorname{vec}(W_i)\cdot \mathbf{1}_{1\times d} \end{bmatrix} \right) - z_i \cdot \mathbf{1}_{1\times d} \right\|_\infty \\
&= \left\| \sum_{h=1}^d \mathrm{Attn}_h^{(1)}\left(\begin{bmatrix} o_{i-1}\cdot \mathbf{1}_{1\times d} \\ \operatorname{vec}(W_i)\cdot \mathbf{1}_{1\times d} \end{bmatrix} \right) - \sum_{h=1}^d (z_i)_h\, e_h^{(d)} \cdot \mathbf{1}_{1\times d} \right\|_\infty \\
&\le \sum_{h=1}^d \left\| \mathrm{Attn}_h^{(1)}\left(\begin{bmatrix} o_{i-1}\cdot \mathbf{1}_{1\times d} \\ \operatorname{vec}(W_i)\cdot \mathbf{1}_{1\times d} \end{bmatrix} \right) - (z_i)_h\, e_h^{(d)} \cdot \mathbf{1}_{1\times d} \right\|_\infty \le d\,\epsilon_1. \qquad \text{((triangle inequality))}
\end{aligned}$$

For convenience, we name the aggregated attention output (for this specific input) as

$$Z_{\mathrm{att}} := \sum_{h=1}^d \mathrm{Attn}_h^{(1)} \circ A\left(\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix} \right), \tag{20}$$

where A is the linear transformation from Step 1, $Z_{\mathrm{att}} \approx z_i \cdot \mathbf{1}_{1\times d}$ and $\|Z_{\mathrm{att}} - z_i \cdot \mathbf{1}_{1\times d}\|_\infty \le d\epsilon_1$.

Step 3: Use a token-wise FFN to produce $\sigma(z_i)$ and $\sigma'(z_i)$. Given $Z_{\mathrm{att}} \approx z_i \cdot \mathbf{1}_{1\times d}$, we next produce

$$o_i \cdot \mathbf{1}_{1\times d} = \sigma(z_i) \cdot \mathbf{1}_{1\times d} \quad \text{and} \quad \sigma'(z_i) \cdot \mathbf{1}_{1\times d},$$

by applying suitable token-wise feed-forward maps to Z_{att}.

We first construct a token-wise feed-forward network FFN_1 dedicated to approximating $\sigma'(\cdot)$:

$$\mathrm{FFN}_1(Z) := \frac{1}{\delta_1}\Big(\sigma(Z) - \sigma(Z - \delta_1\, \mathbf{1}_{d\times d})\Big), \tag{21}$$

where $\delta_1 > 0$ is chosen according to the required precision, Z is a generic input to the FFN, and $\sigma(\cdot)$ is applied entrywise. Equivalently, for each (r, c), (here r indexes feature coordinates and c indexes token positions.)

$$\big(\mathrm{FFN}_1(Z)\big)_{r,c} = \begin{cases} 1, & Z_{r,c} > \delta_1, \\ Z_{r,c}/\delta_1, & 0 < Z_{r,c} \le \delta_1, \\ 0, & Z_{r,c} \le 0. \end{cases}$$

In particular, whenever $Z_{r,c} \notin (0, \delta_1]$ we have $\big(\mathrm{FFN}_1(Z)\big)_{r,c} = \sigma'(Z_{r,c})$. Thus, since the attention module produces $Z_{\mathrm{att}} \approx z_i \cdot \mathbf{1}_{1\times d}$, we obtain

$$\mathrm{FFN}_1(Z_{\mathrm{att}}) \approx \sigma'(z_i) \cdot \mathbf{1}_{1\times d},$$

except on the usual small region where some coordinate lies in $(0, \delta_1]$.

Next, we put the two nonlinear rows inside the same FFN sublayer output via a block stacking FFN:

$$\mathrm{FFN}_2(Z) := \begin{bmatrix} \sigma(Z) \\ 0_{d\times d} \\ \mathrm{FFN}_1(Z) \\ 0_{d^2\times d} \\ \mathbf{1}_{1\times d} \end{bmatrix}, \tag{22}$$

where the last row $\mathbf{1}_{1\times d}$ is produced by biases (with zero input weights). FFN_2 is still an FFN because FFNs are closed under parallelization and post composition with affine maps.

Step 4: Route the remaining rows via a skip connection. The remaining rows $o_{i-1} \cdot \mathbf{1}_{1\times d}$ and $\mathrm{vec}(W_i) \cdot \mathbf{1}_{1\times d}$ are provided by a skip connection from $A(\cdot)$ using a token wise linear map A_1 that places these blocks into the correct output rows:

$$A_1(Z) := \begin{bmatrix} 0_{d\times d} & 0_{d\times d^2} \\ I_d & 0_{d\times d^2} \\ 0_{d\times d} & 0_{d\times d^2} \\ 0_{d^2\times d} & I_{d^2} \\ 0_{1\times d} & 0_{1\times d^2} \end{bmatrix} Z.$$

Step 5: Assemble the transformer block and bound the error. Combining the skip connected linear part and the FFN output from the attention result, the overall single-layer transformer block can be written as

$$\mathcal{F}_{\mathrm{forward}}(Z) \;=\; A_1 \circ A(Z) \;+\; \mathrm{FFN}_2\left(\sum_{h=1}^{d} \mathrm{Attn}_h^{(1)} \circ A(Z)\right). \tag{23}$$

Therefore, on the input $\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix}$ the output equals (approximately)

$$\begin{bmatrix} o_i \cdot \mathbf{1}_{1\times d} \\ o_{i-1} \cdot \mathbf{1}_{1\times d} \\ \sigma'(z_i) \cdot \mathbf{1}_{1\times d} \\ \mathrm{vec}(W_i) \cdot \mathbf{1}_{1\times d} \\ \mathbf{1}_{1\times d} \end{bmatrix}, \qquad \text{where } o_i = \sigma(z_i),$$

except on an arbitrarily small region induced by the approximation in (21).

We now bound the l_∞ error. Since the other rows are matched *exactly* by $A_1 \circ A(\cdot)$ and biases, the l_∞ error reduces to the maximum of the only two nontrivial rows:

$$\begin{aligned}
\left\|\mathcal{F}_{\mathrm{forward}}\left(\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix}\right) - F_i\right\|_\infty &= \max\left\{\left\|\sigma(Z_{\mathrm{att}}) - o_i \cdot \mathbf{1}_{1\times d}\right\|_\infty,\ \left\|\mathrm{FFN}_1(Z_{\mathrm{att}}) - \sigma'(z_i) \cdot \mathbf{1}_{1\times d}\right\|_\infty\right\} \\
&= \max\left\{\left\|\sigma(Z_{\mathrm{att}}) - \sigma(z_i \cdot \mathbf{1}_{1\times d})\right\|_\infty,\ \left\|\mathrm{FFN}_1(Z_{\mathrm{att}}) - \sigma'(z_i) \cdot \mathbf{1}_{1\times d}\right\|_\infty\right\} \\
&\qquad \text{(By Definition 2.2)} \\
&\le \max\left\{\|Z_{\mathrm{att}} - z_i \cdot \mathbf{1}_{1\times d}\|_\infty,\ \left\|\mathrm{FFN}_1(Z_{\mathrm{att}}) - \sigma'(z_i) \cdot \mathbf{1}_{1\times d}\right\|_\infty\right\},
\end{aligned}$$

where the inequality uses the elementary fact that for ReLU, $|\sigma(a) - \sigma(b)| \le |a - b|$ holds entrywise.

By the construction of the d attention heads (Lemma 4.1 applied per head with accuracy ϵ_1), we have

$$\Delta \;:=\; \|Z_{\mathrm{att}} - z_i \cdot \mathbf{1}_{1\times d}\|_\infty \;\le\; d\epsilon_1. \tag{24}$$

Controlling the FFN_1 term via a bad region. Define the bad region

$$\mathcal{B} := \left\{ \begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix} : \; \exists r \in [d] \text{ s.t. } (z_i)_r \in (-\Delta,\; \delta_1 + \Delta] \right\}. \tag{25}$$

We claim that for every input outside $\mathcal{B}$,

$$\mathrm{FFN}_1(Z_{\mathrm{att}}) = \sigma'(z_i) \cdot \mathbf{1}_{1\times d}, \qquad \text{and hence} \qquad \left\|\mathrm{FFN}_1(Z_{\mathrm{att}}) - \sigma'(z_i) \cdot \mathbf{1}_{1\times d}\right\|_\infty = 0. \tag{26}$$

To prove the claim, fix an input outside $\mathcal{B}$ and any entry $(r, c) \in [d] \times [d]$. Since the input is outside $\mathcal{B}$, either $(z_i)_r \le -\Delta$ or $(z_i)_r \ge \delta_1 + \Delta$. In the first case, using (24) we have $(Z_{\mathrm{att}})_{r,c} \le (z_i)_r + \Delta \le 0$, and by the definition of FFN_1 in (21), $(\mathrm{FFN}_1(Z_{\mathrm{att}}))_{r,c} = 0 = \sigma'((z_i)_r)$. In the second case, again by (24) we have $(Z_{\mathrm{att}})_{r,c} \ge (z_i)_r - \Delta \ge \delta_1$, and by (21), $(\mathrm{FFN}_1(Z_{\mathrm{att}}))_{r,c} = 1 = \sigma'((z_i)_r)$.

Since this holds for every (r, c), we conclude (26).

Therefore, for inputs outside $\mathcal{B}$,

$$\begin{aligned} \left\|\mathcal{F}_{\mathrm{forward}}\left(\begin{bmatrix} F_{i-1} \\ B_i \end{bmatrix}\right) - F_i\right\|_\infty &\le \max\left\{\|\sigma(Z_{\mathrm{att}}) - \sigma(z_i \cdot \mathbf{1}_{1\times d})\|_\infty,\; 0\right\} \\ &\le \|Z_{\mathrm{att}} - z_i \cdot \mathbf{1}_{1\times d}\|_\infty \;=\; \Delta \;\le\; d\epsilon_1. \end{aligned}$$

Choosing $\epsilon_1 = \epsilon/d$ yields the desired bound $\le \epsilon$ outside the bad region $\mathcal{B}$. Moreover, recalling $\Delta := \|Z_{\mathrm{att}} - z_i \cdot \mathbf{1}_{1\times d}\|_\infty \le d\epsilon_1$, the definition (25) shows that $\mathcal{B}$ consists exactly of inputs for which some coordinate $(z_i)_r$ lies in the margin interval $(-\Delta,\; \delta_1 + \Delta]$, whose width is $\delta_1 + 2\Delta$. Hence, as $\delta_1 \to 0$ and $\epsilon_1 \to 0$ (so $\Delta \to 0$), this excluded region shrinks to the threshold sets $\{(z_i)_r = 0\}$ and $\{(z_i)_r = \delta_1\}$ in the z_i coordinates. □

C.5 Proof of Theorem 4.2

Theorem C.5 (Restated of Theorem 4.2, In-context Calculation of Backward Propagation)**.** *Let l denote the current round. When $t \bmod 2N \in \{N+1, \cdots, 2N\}$, the network performs backward propagation for the N-layer feed-forward network. The input it receives from the dynamic masking mechanism is*

$$\begin{bmatrix} B_{i+1} \\ F_i \end{bmatrix} = \begin{bmatrix} \delta_{i+1} \cdot 1_{1\times d} \\ W_{i+1}^\top \delta_{i+1} \cdot 1_{1\times d} \\ (\mathrm{vec}(W_{i+1}) - \eta \mathrm{vec}(\delta_{i+1} o_{i+1}^\top)) \cdot 1_{1\times d} \\ 0_{(d+1)\times d} \\ o_i \cdot 1_{1\times d} \\ o_{i-1} \cdot 1_{1\times d} \\ \sigma'(z_i) \cdot 1_{1\times d} \\ \mathrm{vec}(W_i) \cdot 1_{1\times d} \\ 1_{1\times d} \end{bmatrix},$$

For any $\epsilon > 0$, there exists a single-layer transformer network $\mathcal{F}_{\mathrm{backprop}}$ with positional encoding such that

$$\left\| \mathcal{F}_{\mathrm{backprop}}\left(\begin{bmatrix} B_{i+1} \\ F_i \end{bmatrix} \right) - B_i \right\|_\infty \le \epsilon,$$

where The guarantee holds except for an arbitrarily small region in the input space.

$$B_i = \begin{bmatrix} \delta_i \cdot 1_{1\times d} \\ W_i^\top \delta_i \cdot 1_{1\times d} \\ (\mathrm{vec}(W_i) - \eta \mathrm{vec}(\delta_i o_i^\top)) \cdot 1_{1\times d} \\ 0_{(d+1)\times d} \end{bmatrix}.$$

The guarantee holds except for an arbitrarily small region in the input space.

Proof. **Roadmap.** Starting from the masked input $\begin{bmatrix} B_{i+1} \\ F_i \end{bmatrix}$, we proceed in six steps:

1. Extract the specific slots needed for the i-th backprop update.
2. Compute the backprop signal $\delta_i = (W_{i+1}^\top \delta_{i+1}) \odot \sigma'(z_i)$ via a token-wise FFN.
3. Construct the vectorized gradient $\mathrm{vec}(\delta_i o_{i-1}^\top)$ via d attention heads.
4. Update $\mathrm{vec}(W_i)$ by summing a skip branch and a scaled attention branch (implementing gradient descent).
5. Compute $W_i^\top \delta_i$ via another attention branch.
6. Assemble all outputs into B_i and bound the total error.

Step 1: Isolating the terms needed for the i-th backward step: We first apply a token-wise linear map A^* that extracts (and broadcasts across columns) the entries we will use in the subsequent computation:

$$A^*\left(\begin{bmatrix} B_{i+1} \\ F_i \end{bmatrix} \right) = \begin{bmatrix} \delta_{i+1} \cdot 1_{1\times d} \\ W_{i+1}^\top \delta_{i+1} \cdot 1_{1\times d} \\ o_{i-1} \cdot 1_{1\times d} \\ \sigma'(z_i) \cdot 1_{1\times d} \\ \mathrm{vec}(W_i) \cdot 1_{1\times d} \end{bmatrix}.$$

One explicit construction is the following matrix:

$$A^*(Z) := \begin{bmatrix} I_d & 0_{d\times d} & 0_{d\times(d^2+2d+1)} & 0_{d\times(2d+d^2)} \\ 0_{d\times d} & I_d & 0_{d\times(d^2+2d+1)} & 0_{d\times(2d+d^2)} \\ 0_{(2d+d^2)\times d} & 0_{(2d+d^2)\times d} & 0_{(2d+d^2)\times(d^2+2d+1)} & I_{2d+d^2} \end{bmatrix} Z.$$

Step 2: Computing the ReLU backprop term δ_i: We now construct a token-wise feed-forward network that computes

$$\delta_i \;=\; \big(W_{i+1}^\top \delta_{i+1}\big) \odot \sigma'(z_i) \in \mathbb{R}^d$$

and broadcasts it as $\delta_i \cdot 1_{1\times d}$. Outside an arbitrarily small region of the input space where some coordinate of z_i lies in the transition interval of σ', we have $\sigma'(z_i) \in \{0,1\}^d$. Fix any constant

$$B_1 \;\geq\; \big\|W_{i+1}^\top \delta_{i+1}\big\|_\infty.$$

For each coordinate index $r \in [d]$, we aim to output

$$(\delta_i)_r \;=\; \big(W_{i+1}^\top \delta_{i+1}\big)_r \cdot \sigma'(z_i)_r.$$

Define the token wise FFN $\mathrm{FFN}_3 : \mathbb{R}^{(4d+d^2)\times d} \to \mathbb{R}^{d\times d}$ by

$$\mathrm{FFN}_3(Z) := \sigma\Big(\begin{bmatrix}0_{d\times d} & I_d & 0_{d\times d} & 2B_1 I_d & 0_{d\times d^2}\end{bmatrix} Z - B_1 \cdot 1_{d\times d}\Big) - B_1 \cdot \begin{bmatrix}0_{d\times d} & 0_{d\times d} & 0_{d\times d} & I_d & 0_{d\times d^2}\end{bmatrix} Z,$$

where $\sigma(\cdot)$ is ReLU applied entrywise.

Since each extracted vector is broadcast across columns, the output of $\mathrm{FFN}_3 \circ A^*(\cdot)$ has identical columns. Substituting the output of A^* gives

$$\mathrm{FFN}_3 \circ A^*\left(\begin{bmatrix}B_{i+1}\\ F_i\end{bmatrix}\right) = \Big(\sigma\big(W_{i+1}^\top \delta_{i+1} + 2B_1\sigma'(z_i) - B_1\cdot 1_d\big) - B_1\sigma'(z_i)\Big)\cdot 1_{1\times d}.$$

Fix $r \in [d]$. Since $B_1 \geq \|W_{i+1}^\top \delta_{i+1}\|_\infty$, we have $\big(W_{i+1}^\top \delta_{i+1}\big)_r \in [-B_1, B_1]$. For $\sigma'(z_i)_r \in \{0,1\}$,

$$\Big[\sigma\big(W_{i+1}^\top \delta_{i+1} + 2B_1\sigma'(z_i) - B_1\cdot 1_d\big) - B_1\sigma'(z_i)\Big]_r = \begin{cases}\sigma\big((W_{i+1}^\top \delta_{i+1})_r + B_1\big) - B_1 = (W_{i+1}^\top \delta_{i+1})_r, & \sigma'(z_i)_r = 1,\\ \sigma\big((W_{i+1}^\top \delta_{i+1})_r - B_1\big) - 0 = 0, & \sigma'(z_i)_r = 0,\end{cases}$$

which equals $\big(W_{i+1}^\top \delta_{i+1}\big)_r \cdot \sigma'(z_i)_r$ in both cases. Therefore,

$$\mathrm{FFN}_3 \circ A^*\left(\begin{bmatrix}B_{i+1}\\ F_i\end{bmatrix}\right) = \big((W_{i+1}^\top \delta_{i+1}) \odot \sigma'(z_i)\big)\cdot 1_{1\times d} = \delta_i \cdot 1_{1\times d}.$$

Step 3: Building the gradient block $\mathrm{vec}(\delta_i o_{i-1}^\top)$: We next construct $\mathrm{vec}(\delta_i o_{i-1}^\top)\cdot 1_{1\times d}$ from $\delta_i \cdot 1_{1\times d}$ and $o_{i-1}\cdot 1_{1\times d}$. Fix any $\epsilon_2 > 0$. By Lemma A.5, for each $h \in [d]$ there exists a multi-head attention module Attn_h such that

$$\left\|\mathrm{Attn}_h\left(\begin{bmatrix}\delta_i \cdot 1_{1\times d}\\ o_{i-1}\cdot 1_{1\times d}\end{bmatrix}\right) - \begin{bmatrix}0_{(h+1)d\times d}\\ (o_{i-1})_h\,\delta_i\cdot 1_{1\times d}\\ 0_{((d-h+1)d+1)\times d}\end{bmatrix}\right\|_\infty \leq \epsilon_2.$$

Summing over h and using the triangle inequality yields

$$\left\|\sum_{h=1}^d \mathrm{Attn}_h\left(\begin{bmatrix}\delta_i \cdot 1_{1\times d}\\ o_{i-1}\cdot 1_{1\times d}\end{bmatrix}\right) - \sum_{h=1}^d \begin{bmatrix}0_{(h+1)d\times d}\\ (o_{i-1})_h\,\delta_i\cdot 1_{1\times d}\\ 0_{((d-h+1)d+1)\times d}\end{bmatrix}\right\|_\infty \leq d\epsilon_2.$$

To identify the deterministic sum with $\mathrm{vec}(\delta_i o_{i-1}^\top)$, recall the column-stacking convention

$$\mathrm{vec}(M) := (M_{1,1},\ldots,M_{d,1},\, M_{1,2},\ldots,M_{d,2},\,\ldots,\,M_{1,d},\ldots,M_{d,d})^\top, \qquad M \in \mathbb{R}^{d\times d}.$$

Since

$$\delta_i o_{i-1}^\top = \big[(o_{i-1})_1\delta_i, \ldots, (o_{i-1})_d\delta_i\big],$$

the h-th length-d block of $\mathrm{vec}(\delta_i o_{i-1}^\top)$ equals $(o_{i-1})_h\delta_i$. Hence,

$$\sum_{h=1}^{d} \begin{bmatrix} 0_{(h+1)d\times d} \\ (o_{i-1})_h\, \delta_i \cdot 1_{1\times d} \\ 0_{((d-h+1)d+1)\times d} \end{bmatrix} = \begin{bmatrix} 0_{2d\times d} \\ \mathrm{vec}(\delta_i o_{i-1}^\top)\cdot 1_{1\times d} \\ 0_{(d+1)\times d} \end{bmatrix},$$

and therefore

$$\left\| \sum_{h=1}^{d} \mathrm{Attn}_h\left(\begin{bmatrix} \delta_i \cdot 1_{1\times d} \\ o_{i-1}\cdot 1_{1\times d} \end{bmatrix} \right) - \begin{bmatrix} 0_{2d\times d} \\ \mathrm{vec}(\delta_i o_{i-1}^\top)\cdot 1_{1\times d} \\ 0_{(d+1)\times d} \end{bmatrix} \right\|_\infty \le d\epsilon_2.$$

Step 4: Implement the gradient descent update on $\mathrm{vec}(W_i)$: A transformer block outputs the sum of its branches, and token-wise linear maps can scale their outputs arbitrarily. Thus we implement the update in the $\mathrm{vec}(W_i)$-slot by summing: (i) a token-wise skip branch that places $\mathrm{vec}(W_i)\cdot 1_{1\times d}$ into the $\mathrm{vec}(W_i)$ rows, and (ii) the attention branch $\sum_{h=1}^{d}\mathrm{Attn}_h(\cdot)$ from Step 3, scaled by $-\eta$. This yields (in the intended rows)

$$\mathrm{vec}(W_i)\cdot 1_{1\times d} + \big(-\eta\,\mathrm{vec}(\delta_i o_{i-1}^\top)\big)\cdot 1_{1\times d} = \big(\mathrm{vec}(W_i) - \eta\,\mathrm{vec}(\delta_i o_{i-1}^\top)\big)\cdot 1_{1\times d}.$$

Note that we realize the factor $-\eta$ by absorbing it into the attention head's value, since $\mathrm{Attn}(Q,K,cV) = c\,\mathrm{Attn}(Q,K,V)$.

Step 5: Compute $W_i^\top\delta_i$ for the next backward signal: Fix any $\epsilon_3 > 0$. By Lemma A.5, for each $h\in[d]$ there exists a multi-head attention module Attn_h^* such that

$$\left\| \mathrm{Attn}_h^*\left(\begin{bmatrix} \delta_i\cdot 1_{1\times d} \\ \mathrm{vec}((W_i)_{h,:})\cdot 1_{1\times d} \end{bmatrix} \right) - (W_i)_{h,:}^\top\delta_i \cdot e_{h+d}^{(3d+d^2+1)}\cdot 1_{1\times d} \right\|_\infty \le \epsilon_3.$$

Summing over h places $W_i^\top\delta_i\cdot 1_{1\times d}$ into the designated coordinates (with total l_∞ error at most $d\epsilon_3$).

Step 6: Assemble a single transformer block and bound: We now combine (a) the gradient term from Step 3 (scaled by $-\eta$), (b) the dot-product term from Step 5, and (c) a skip/placement branch that carries δ_i and $\mathrm{vec}(W_i)$. Define

$$\mathcal{F}_{\mathrm{backprop}}(Z) := -\eta\sum_{h=1}^{d}\mathrm{Attn}_h\left(\begin{bmatrix}\mathrm{FFN}_3\circ A^*(Z)\\ EA^*(Z)\end{bmatrix}\right) + \sum_{h=1}^{d}\mathrm{Attn}_h^*\left(\begin{bmatrix}\mathrm{FFN}_3\circ A^*(Z)\\ E_hA^*(Z)\end{bmatrix}\right) + \left(\begin{bmatrix} I_d \\ 0_{(2d+d^2+1)\times d}\end{bmatrix}\mathrm{FFN}_3 + A_s\right)\circ A^*(Z), \tag{27}$$

where the token-wise selectors are

$$E := \begin{bmatrix} I_d & 0_{d\times(d^2+4d)}\end{bmatrix}, \qquad E_h := \begin{bmatrix} 0_{d\times(h+3)d} & I_d & 0_{d\times(d-h)d}\end{bmatrix}.$$

informing and the skip/placement map is

$$A_s(Z) := \begin{bmatrix} 0_{2d\times 4d} & 0_{2d\times d^2} \\ 0_{d^2\times 4d} & I_{d^2} \\ 0_{(d+1)\times 4d} & 0_{(d+1)\times d^2}\end{bmatrix} Z.$$

Define token-wise linear padding maps $P_{\mathrm{top}}u := \begin{bmatrix}u\\0\end{bmatrix}$ and $P_{\mathrm{bot}}v := \begin{bmatrix}0\\v\end{bmatrix}$, so that $P_{\mathrm{top}}\big(\mathrm{FFN}_3(A^*(Z))\big) + P_{\mathrm{bot}}\big(EA^*(Z)\big) = \begin{bmatrix}\mathrm{FFN}_3(A^*(Z))\\ EA^*(Z)\end{bmatrix}$. All maps $A^*, E, E_h, A_s, P_{\mathrm{top}}, P_{\mathrm{bot}}$ are fixed token-wise linear maps ($Z\mapsto TZ$). Such that any T can be absorbed into standard token wise projections. Now, for attention, replace W_Q, W_K, W_V by W_QT, W_KT, W_VT; for an FFN $\mathrm{FFN}(Z) = W_2\sigma(W_1Z)$, replace W_1 by W_1T. Hence the terms $EA^*(Z)$ and $E_hA^*(Z)$ in (27) are realized by standard attention/FFN layers with modified projections, and $P_{\mathrm{top}}, P_{\mathrm{bot}}, A_s$ are standard token-wise padding/skip maps, so (27) uses only standard Transformer components.

Finally, the constructions in Steps 2–5 show that each required block of B_i is produced (in the appropriate slots) up to errors

$d\epsilon_2$ and $d\epsilon_3$ coming from Lemma A.5. Thus, for any target accuracy $\varepsilon > 0$, choosing $\epsilon_2 \leq \varepsilon/(2\eta d)$ and $\epsilon_3 \leq \varepsilon/(2d)$ gives

$$\left\| \mathcal{F}_{\text{backprop}} \left(\begin{bmatrix} B_{i+1} \\ F_i \end{bmatrix} \right) - B_i \right\|_\infty \leq \varepsilon,$$

except for an arbitrarily small input region corresponding to the transition interval of σ'. Since $\varepsilon > 0$ is arbitrary, this completes the proof. □

C.6 Proof of Proposition 4.1

For the special occasion when X_s and Y_s for $s \in [n]$ are given in the input, we also devise the following proposition to calculate the forward/ backward step for this input.

Proposition C.1 (Proposition 4.1 Restated: Initialization of the First Forward Block)**.** *Fix $s \in [n]$ where n is the number of in-context examples, and let X_s, B_1, and F_1 be the blocks defined in Definition 4.3. Then for any $\varepsilon > 0$, there exists a single Transformer block (with residual/skip connections) $\mathcal{F}_x$ such that, for the input obtained by concatenating the two blocks,*

$$Z = \begin{bmatrix} B_1 \\ X_s \end{bmatrix},$$

the output $\mathcal{F}_x(Z)$ approximates the desired forward block F_1 such that

$$\left\| \mathcal{F}_x(Z) - F_1 \right\|_\infty \leq \varepsilon.$$

Proof. Let A_3 be

$$A_3(Z) := \begin{bmatrix} \begin{bmatrix} 0_{d\times(1+3d+d^2)} & I_d & 0_{d\times(2d+d^2+1)} \end{bmatrix} Z \\ \begin{bmatrix} 0_{d^2\times 2d} & I_{d^2} & 0_{d^2\times(d+1)} \end{bmatrix} Z \end{bmatrix}.$$

This outputs

$$A_3(\begin{bmatrix} B_1 \\ X_s \end{bmatrix}) = \begin{bmatrix} x_s \cdot 1_{1\times d} \\ \mathrm{vec}(W_1) \cdot 1_{1\times d} \end{bmatrix}.$$

Now according to Lemma A.5, for every $h \in [d]$, there is a multi-head attention $\mathrm{Attn}_h^{(2)}$ such that

$$\| \sum_{h=1}^{d} \mathrm{Attn}_h^{(2)}(\begin{bmatrix} x_s \cdot 1_{1\times d} \\ \mathrm{vec}(W_1) \cdot 1_{1\times d} \end{bmatrix}) - z_1 \|_\infty \leq \epsilon_4$$

for any $\epsilon_4 > 0$. This output then goes through a ReLU function and goes into the final output.

Then use a FFN_4 constructed as FFN_1 in Section C.4 satisfying

$$\mathrm{FFN}_4(z_1) = \sigma'(z_1).$$

Append FFN_4 to the output of $\sum_{h=1}^{d} \mathrm{Attn}_h^{(2)}$ outputs $\sigma'(z_1)1_{1\times d}$.

Concatenating the outputs of $\sum_{h=1}^{d} \mathrm{Attn}_h^{(2)}$ and FFN_4 with $x_s 1_{1\times d}$ and $\mathrm{vec}W_1 1_{1\times d}$ in the input yields

$$\begin{bmatrix} \sigma(\sum_{h=1}^{d} \mathrm{Attn}_h^{(2)}(\begin{bmatrix} x_s \cdot 1_{1\times d} \\ \mathrm{vec}(W_1) \cdot 1_{1\times d} \end{bmatrix})) \\ x_s \cdot 1_{1\times d} \\ \sigma'(z_1) \cdot 1_{1\times d} \\ \mathrm{vec}(W_1) \cdot 1_{1\times d} \\ 1_{1\times d}, \end{bmatrix}.$$

That's closer to F_1 by ϵ_4 in infinite norm. □

C.7 Proof of Proposition 4.2

Proposition C.2 (Proposition 4.2 Restated: Initialization of the Backward Block)**.** *Fix $s \in [n]$, where n is the number of in-context examples, and let F_N, Y_s, and B_N be the blocks defined in Definition 4.3. Then for any $\epsilon > 0$, there exists a single Transformer block (with residual/skip connections) $\mathcal{F}_y$ such that, for the input obtained by concatenating the two blocks,*

$$Z = \begin{bmatrix} F_N \\ Y_s \end{bmatrix},$$

the output $\mathcal{F}_y(Z)$ approximates the desired backward block B_N such that

$$\|\mathcal{F}_y(Z) - B_N\|_\infty \leq \epsilon.$$

Proof. First we use a token-wise linear transformation A_4 to extract the needed rows from the input.

$$A_4(\begin{bmatrix} F_N \\ Y_s \end{bmatrix}) := \begin{bmatrix} o_N \cdot 1_{1\times d} \\ y_s \cdot 1_{1\times d} \\ \mathrm{vec}(W_N) \cdot 1_{1\times d} \end{bmatrix}.$$

Then, there exists a feed-forward network FFN_5 such that

$$\mathrm{FFN}_5(o_N) = \sigma'(o_N) = \sigma'(z_N)$$

except for an arbitrarily small region.

Same as in Section C.5 by Lemma A.5, there is a set of d multi-head attention, whose sum we note by a larger multi-head attention $\mathrm{Attn}_m^{(1)}$ that satisfies

$$\|\mathrm{Attn}_m^{(1)}(\begin{bmatrix} o_N \cdot 1_{1\times d} \\ y_s \cdot 1_{1\times d} \\ \mathrm{vec}(W_N) \cdot 1_{1\times d} \end{bmatrix}) - \mathrm{vec}(\delta_N o_N^\top) \cdot 1_{1\times d}\|_\infty \leq \epsilon_5$$

for any $\epsilon_5 > 0$.

Finally, upon the output of FFN_5, same as in Section C.5, there is an FFN_6 that calculates

$$\underbrace{2(y_s - o_N) \odot \sigma'(z_N)}_{\delta_N}.$$

Concatenating the above result with token-wise linear transformations yields

$$\begin{bmatrix} \delta_N \cdot 1_{1\times d} \\ W_N^\top \delta_N \cdot 1_{1\times d} \\ \mathrm{vec}(W_N) - \eta \mathrm{Attn}_m^{(1)}(\begin{bmatrix} o_N \cdot 1_{1\times d} \\ y_s \cdot 1_{1\times d} \\ \mathrm{vec}(W_N) \cdot 1_{1\times d} \end{bmatrix} \cdot 1_{1\times d}) \\ 0_{(d+1)\times d} \end{bmatrix},$$

whose difference with B_N is bounded by ϵ_5. □

C.8 Proof of Proposition 4.3

- 1. Construct a router that assembles the component sub-nets
- 2. Analyze the error accumulated across different steps.

First have a theorem that make a network that executes every step under a certain error, and these steps are continuous.

Proposition C.3 (Proposition 4.3 Restated: Universal One-Step Update Block)**.** *Fix $\varepsilon > 0$. There exists a single-layer Transformer block* $\mathrm{TF} : \mathbb{R}^{D\times d} \to \mathbb{R}^{D\times d}$ *with the following property.*

Let $\mathcal{Z}$ denote the set of valid routed inputs $Z \in \mathbb{R}^{D\times d}$ that arise from the block formats in Definition 4.3, i.e., Z is exactly one of the four types: forward-type*,* backward-type*, x-*type*, or y-*type*. Let $M > 0$ be any constant such that $\|Z\|_\infty \le M/2$ for all $Z \in \mathcal{Z}$.*

Then for every $Z \in \mathcal{Z}$,

$$\|\mathrm{TF}(Z) - T(Z)\|_\infty \le \varepsilon,$$

where T is the corresponding target one-step update map:

$$T(Z) = \begin{cases} T_{\mathrm{fwd}}(Z), & Z \text{ is forward-type}, \\ T_{\mathrm{bwd}}(Z), & Z \text{ is backward-type}, \\ T_x(Z), & Z \text{ is } x\text{-type}, \\ T_y(Z), & Z \text{ is } y\text{-type}. \end{cases}$$

Proof. **Goal.** Construct a *single* Transformer block that, on any input block Z, automatically selects the correct sub computation (forward, backward, x update, or y update) and returns the corresponding next block.

Roadmap. *Step 1:* Use two indicator rows in each block to build four gates. *Step 2:* Prove each gate either passes Z through unchanged or zeros out. *Step 3:* Run all four subnetworks in parallel and sum; since exactly one gate is active, the sum equals the desired output.

Step 1: Indicator rows and gated affine maps. Let $Z \in \mathbb{R}^{D\times d}$ be an input block (each column is a token embedding). Let $r_u := d^2 + 3d + 1$ and $r_v := 2d^2 + 6d + 2$, and define the row extractors

$$u(Z) := Z_{r_u,:} \in \mathbb{R}^{1\times d}, \qquad v(Z) := Z_{r_v,:} \in \mathbb{R}^{1\times d}.$$

By the block design of F_i, B_i, X_s, Y_s, every Z satisfies *exactly one* of the tag identities (entrywise across the d columns):

$$\begin{array}{ll} \text{(forward-type)} & u(Z) - v(Z) = \mathbf{1}_{1\times d}, \\ \text{(backward-type)} & -u(Z) - v(Z) + \mathbf{1}_{1\times d} = 0, \\ (x\text{-type}) & -u(Z) + v(Z) = \mathbf{1}_{1\times d}, \\ (y\text{-type}) & u(Z) + v(Z) = 3\,\mathbf{1}_{1\times d}. \end{array}$$

Fix $M > 0$ such that for every block Z we have $\|Z\|_\infty \le M/2$. Define the shifted ReLU gate (entrywise)

$$\sigma_0(W) := \mathrm{ReLU}\Big(W + \frac{M}{2}\Big) - \frac{M}{2}.$$

Let $\mathbf{1}_{D\times 1}$ denote the all-ones column vector. Define four token-wise affine maps

$$\begin{aligned} K_f(Z) &:= Z + M\,\mathbf{1}_{D\times 1}\big(u(Z) - v(Z) - \mathbf{1}_{1\times d}\big), \\ K_b(Z) &:= Z + M\,\mathbf{1}_{D\times 1}\big(-u(Z) - v(Z) + \mathbf{1}_{1\times d}\big), \\ K_x(Z) &:= Z + M\,\mathbf{1}_{D\times 1}\big(-u(Z) + v(Z) - \mathbf{1}_{1\times d}\big), \\ K_y(Z) &:= Z + M\,\mathbf{1}_{D\times 1}\big(u(Z) + v(Z) - 3\mathbf{1}_{1\times d}\big). \end{aligned}$$

Finally define the router output (stacking four $D \times d$ blocks):

$$R(Z) := \begin{bmatrix} \sigma_0(K_f(Z)) \\ \sigma_0(K_b(Z)) \\ \sigma_0(K_x(Z)) \\ \sigma_0(K_y(Z)) \end{bmatrix} \in \mathbb{R}^{4D \times d}.$$

Step 2: Gating property. We record the key fact once; the other cases are identical. If Z is forward type, then $u(Z) - v(Z) - \mathbf{1}_{1\times d} = 0$, hence $K_f(Z) = Z$, so every entry of $K_f(Z)$ lies in $[-M/2, M/2]$ and therefore $\sigma_0(K_f(Z)) = Z$. If Z is *not* forward type, then each entry of $u(Z) - v(Z) - \mathbf{1}_{1\times d}$ is ≤ -1, hence

$$K_f(Z) \;\leq\; Z - M\mathbf{1}_{D\times d} \;\leq\; -\frac{M}{2}\mathbf{1}_{D\times d},$$

so $\sigma_0(K_f(Z)) = 0$. Thus,

$$\sigma_0(K_f(Z)) = \begin{cases} Z, & Z \text{ is forward-type,} \\ 0, & \text{otherwise.} \end{cases}$$

Similarly,

$$\sigma_0(K_b(Z)) = \begin{cases} Z, & Z \text{ is backward-type,} \\ 0, & \text{otherwise,} \end{cases} \quad \sigma_0(K_x(Z)) = \begin{cases} Z, & Z \text{ is } x\text{-type,} \\ 0, & \text{otherwise,} \end{cases} \quad \sigma_0(K_y(Z)) = \begin{cases} Z, & Z \text{ is } y\text{-type,} \\ 0, & \text{otherwise.} \end{cases}$$

Since the four tag identities are mutually exclusive, exactly one of these four gated outputs equals Z and the other three are 0.

Step 3: Combining the updates into a single block Fix $\varepsilon > 0$. Let $\mathcal{F}_{\text{fwd}}, \mathcal{F}_{\text{bwd}}, \mathcal{F}_x, \mathcal{F}_y : \mathbb{R}^{D\times d} \to \mathbb{R}^{D\times d}$ denote the (single block) Transformer computations from Theorem 4.1, Theorem 4.2, Proposition C.1, and Proposition C.2, respectively, chosen so that each approximates its corresponding target update map to ε-accuracy in $\|\cdot\|_\infty$

Form their *direct sum* Transformer $\mathcal{F}_\oplus : \mathbb{R}^{4D\times d} \to \mathbb{R}^{4D\times d}$ that applies the four maps to four stacked $D \times d$ blocks:

$$\mathcal{F}_\oplus\left(\begin{bmatrix} Z_f \\ Z_b \\ Z_x \\ Z_y \end{bmatrix}\right) := \begin{bmatrix} \mathcal{F}_{\text{fwd}}(Z_f) \\ \mathcal{F}_{\text{bwd}}(Z_b) \\ \mathcal{F}_x(Z_x) \\ \mathcal{F}_y(Z_y) \end{bmatrix}.$$

Let $A : \mathbb{R}^{4D\times d} \to \mathbb{R}^{D\times d}$ be the linear aggregation map

$$A\left(\begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{bmatrix}\right) := W_1 + W_2 + W_3 + W_4,$$

Define the final single-block mechanism by

$$\mathrm{TF}(Z) \;:=\; A \circ \mathcal{F}_\oplus \circ R(Z).$$

Now fix an input Z. By Step 2, $R(Z)$ is one-hot in the sense that it equals $[Z; 0; 0; 0]$ (forward-type), or $[0; Z; 0; 0]$ (backward-type), or $[0; 0; Z; 0]$ (x-type), or $[0; 0; 0; Z]$ (y-type). Therefore,

$$\mathrm{TF}(Z) = \begin{cases} \mathcal{F}_{\text{fwd}}(Z), & Z \text{ forward-type,} \\ \mathcal{F}_{\text{bwd}}(Z), & Z \text{ backward-type,} \\ \mathcal{F}_x(Z), & Z\ x\text{-type,} \\ \mathcal{F}_y(Z), & Z\ y\text{-type.} \end{cases}$$

Let $\mathcal{B}$ denote the union of the (arbitrarily small) bad regions associated with the four submodules and the router. For any $Z \in \mathcal{Z} \setminus \mathcal{B}$, we have

$$\|\mathrm{TF}(Z) - T(Z)\|_\infty \leq \varepsilon.$$

If Z is forward-type then $T(Z) = T_{\mathrm{fwd}}(Z)$ and $\mathrm{TF}(Z) = \mathcal{F}_{\mathrm{fwd}}(Z)$, so $\|\mathrm{TF}(Z) - T(Z)\|_\infty = \|\mathcal{F}_{\mathrm{fwd}}(Z) - T_{\mathrm{fwd}}(Z)\|_\infty \leq \varepsilon$. The other three cases are identical.

Finally, since R and A are compositions of token-wise affine maps and ReLU, and $\mathcal{F}_\oplus$ is obtained by block-diagonal (parallel) composition of single-block Transformers, the overall map $\mathrm{TF} := A \circ \mathcal{F}_\oplus \circ R$ is itself a (single-layer) Transformer block with routing/dynamic masking, as claimed. □

C.9 Proof of Proposition 4.4

Proposition C.4 (In-context GD State Trajectory via Single Transformer Block)**.** *Fix integers $N \geq 1$ and $L \geq 1$, and a labeled sequence $(x_l, y_l)_{l=1}^{L}$ with input formatting as above. Let $\varepsilon_{\mathrm{C1}}, \varepsilon_{4.1}, \varepsilon_{\mathrm{C2}}, \varepsilon_{4.2} > 0$ be the accuracy parameters from Proposition 4.1,Theorem 4.1, Proposition 4.2, and Theorem 4.2, and let $\varepsilon_{\mathrm{C3}} > 0$ be the routing accuracy parameter from Proposition 4.3 . Define*

$$\varepsilon_{\mathrm{round}} := \varepsilon_{\mathrm{C1}} + (N-1)\varepsilon_{4.1} + \varepsilon_{\mathrm{C2}} + (N-1)\varepsilon_{4.2} + (2N+1)\varepsilon_{\mathrm{C3}}.$$

Then there exists a single-layer Transformer block TF *defined in Proposition 4.3, such that it produces state blocks $\{\widehat{B}_i^{(l)}\}_{i\in[N],\,\ell=1}^{L+1}$ satisfying $\widehat{B}_i^{(1)} = B_i^{(1)}$ and, for every $l \in [L]$,*

$$\max_{i\in[N]} \left\|\widehat{B}_i^{(\ell+1)} - B_i^{(\ell+1)}\right\|_\infty \;\leq\; \ell\,\varepsilon_{\mathrm{round}}.$$

In particular,

$$\max_{i\in[N]} \left\|\widehat{B}_i^{(L+1)} - B_i^{(L+1)}\right\|_\infty \;\leq\; L\,\varepsilon_{\mathrm{round}}.$$

Proof. **Proof outline.** We prove the claimed L-round state error bound in three steps: (i) show that, for a *fixed* round l, one full forward–backward sweep implemented by TF incurs at most $\varepsilon_{\mathrm{round}}$ additional $\|\cdot\|_\infty$ error in the resulting state blocks; (ii) show the *exact* round-l update map is non expansive (iii) combine (i)–(ii) to obtain the recursion $E_{l+1} \leq E_l + \varepsilon_{\mathrm{round}}$ and unroll it.

Step 0 (norm and notation). For a stack of state blocks $B^{(l)} := [\,B_1^{(l)} \;\cdots\; B_N^{(l)}\,]$, define

$$\|B^{(l)}\|_{\infty,\max} := \max_{i\in[N]} \|B_i^{(l)}\|_\infty, \qquad E_l := \|\widehat{B}^{(l)} - B^{(l)}\|_{\infty,\max}.$$

By exact initialization, $E_1 = 0$.

Step 1 (one-round approximation error is $\leq \varepsilon_{\mathrm{round}}$). Fix an arbitrary round $l \in [L]$. Let $\mathcal{U}_\ell$ denote the exact round-l update map $[B_1^{(l)}, \ldots, B_N^{(l)}] \mapsto [B_1^{(l+1)}, \ldots, B_N^{(l+1)}]$, and let $\widehat{\mathcal{U}}_l$ be the corresponding map implemented by TF with routing.

By Proposition C.3, each invocation of TF on an input executes the intended sub computation (Prop. C.1 / Thm. 4.1 / Prop. C.2 / Thm. 4.2) with an additional $\|\cdot\|_\infty$ error at most $\varepsilon_{\mathrm{C3}}$. Combining this routing error with the approximation errors $\varepsilon_{\mathrm{C1}}, \varepsilon_{4.1}, \varepsilon_{\mathrm{C2}}, \varepsilon_{4.2}$ accrued across the N forward steps and N backward steps yields the per-round bound

$$\left\|\widehat{\mathcal{U}}_l(B^{(l)}) - \mathcal{U}_l(B^{(l)})\right\|_{\infty,\max} \;\leq\; \varepsilon_{\mathrm{round}},$$

where

$$\varepsilon_{\mathrm{round}} = \varepsilon_{\mathrm{C1}} + (N-1)\varepsilon_{4.1} + \varepsilon_{\mathrm{C2}} + (N-1)\varepsilon_{4.2} + (2N+1)\varepsilon_{\mathrm{C3}}.$$

Step 2 (the exact update map is non expansive). We claim that for every $l \in [L]$ and any two state stacks B, B',

$$\left\|\mathcal{U}_l(B) - \mathcal{U}_l(B')\right\|_{\infty,\max} \;\leq\; \|B - B'\|_{\infty,\max}.$$

This follows because $\mathcal{U}_l$ is obtained by composing the layer-wise forward and backward maps from Theorems 4.1–4.2, and those maps are 1-Lipschitz in $\|\cdot\|_\infty$ in the relevant arguments; taking the maximum over layers preserves the 1-Lipschitz constant.

Step 3 (one-step recursion). Now, notice that

$$\begin{aligned}
E_{l+1} &= \left\|\widehat{B}^{(l+1)} - B^{(l+1)}\right\|_{\infty,\max} \\
&= \left\|\widehat{\mathcal{U}}_l(\widehat{B}^{(l)}) - \mathcal{U}_l(B^{(l)})\right\|_{\infty,\max} \\
&\leq \left\|\widehat{\mathcal{U}}_l(\widehat{B}^{(l)}) - \mathcal{U}_l(\widehat{B}^{(l)})\right\|_{\infty,\max} + \left\|\mathcal{U}_l(\widehat{B}^{(l)}) - \mathcal{U}_l(B^{(l)})\right\|_{\infty,\max} \qquad \text{(triangle inequality)}
\end{aligned}$$

$$\leq \varepsilon_{\text{round}} + \left\|\widehat{B}^{(l)} - B^{(l)}\right\|_{\infty,\max} \qquad \text{(Step 1 + Step 2 (non-expansiveness))}$$
$$= \varepsilon_{\text{round}} + E_l.$$

Unrolling this recursion using $E_1 = 0$ yields $E_{l+1} \leq l\,\varepsilon_{\text{round}}$ for all $l \in [L]$. Taking $l = L$ gives $E_{L+1} \leq L\,\varepsilon_{\text{round}}$.

□

D Experimental Studies

Objectives. The goal of the experiment section is to demonstrate single single-layer decoder-only transformer can simulate multi-step ICGD process for N-layer neural networks by generating CoT sequence.

We aim to train a single-layer autoregressive decoder-only transformer, augmented with the proposed dynamic masking, to generate the CoT sequence that simulates the multi-step ICGD process for a variety of N-layer neural network.

In first step we train the forward operator and backward operator seperatly. For forward operator, the input data is in the form of Theorem 4.1. Formally, the training dataset for forward transformer block $\mathcal{F}_{\text{forward}}$ for one teacher neural network is $\mathcal{X}^{k,\text{forward}} := \{X_i^{k,\text{forward}}, Y_i^{k,\text{forward}}\}_{1\le i\le N}$ where each data pairs is

$$X_i^{k,\text{forward}} = \begin{bmatrix} F_{i-1}^k \\ B_i^k \end{bmatrix}, \quad Y_i^{k,\text{forward}} = F_i^k.$$

We hope by this training data, once the forward transformer block $\mathcal{F}_{\text{forward}}$ see the input selected by dynamic masking, it output the desired next-step forward blocks. In our theory, the transformer blocks can perform the same forward calculation no matter which neural network we want to simulate its forward calculation path. Hence in the training data, we also create K different neural networks training data. That is $\mathcal{X}^{\text{forward}} = \{\mathcal{X}^{k,\text{forward}}\}_{1\le k\le K}$.

Similarly, for backward operator, we form the training dataset as in Theorem 4.2 and the above logic to train a backward operator transformer blocks $\mathcal{F}_{\text{backprop}}$. $\mathcal{X}^{k,\text{backprop}} := \{X_i^{\text{backprop}}, Y_i^{\text{backprop}}\}_{1\le i\le N}$ where each datapoint is

$$X_i^{k,\text{backprop}} = \begin{bmatrix} B_{i+1}^k \\ F_i^k \end{bmatrix}, \quad Y_i^{k,\text{backprop}} = B_i^k.$$

For simplicity, we set $d = 5$, start from $N = 3$, and set total GD updates step to be $L = 20$.

Model Architecture. We use a one-layer transformer block, containing one multi-head attention and one FFN layer. We set the output dimension of the FFN layer to match the dimension of column vector in F_i. The number of head we test starting from $h = 8$.

Data Generation. We set the hidden dimension of the simulated neural network with $d \in \{10, 20\}$, and the number of layer $N \in \{3, 6, 9\}$.

Training Objective. During training we sample mini batches that mix all teachers' neural networks $k \in [K]$, GD rounds $\ell \in [L]$, and hidden layers $i \in [N]$. Given a batch of B pairs (X_b, Y_b), where each $X_b \in \mathbb{R}^{d\times R}$ and $Y_b \in \mathbb{R}^{d\times R}$ (with $R = 1 + 3d + d^2$), we minimise the mean squared Frobenius error

$$\mathcal{L}_{\text{forward}} = \frac{1}{B}\sum_{b=1}^{B} \|\mathcal{F}_{\text{forward}}(X_b^{\text{forward}}) - Y_b^{\text{forward}}\|_F^2.$$

We use the same train loss for the backward operator. Thus, every parameter update encourages the model to reproduce all entries of the next block, and the random batching forces the transformers to generalize across different layers i, optimization steps ℓ, and teacher networks k.

Evaluation and Metrics. We evaluate whether the trained single-layer transformer blocks ($\mathcal{F}_{\text{forward}}$ and $\mathcal{F}_{\text{backprop}}$) simulate a full $2NL$-step ICGD trajectory. Specifically, we initialize with the input-output pair (X_s, Y_s), and then repeatedly apply these two learned operators (by using dynamic masking to select appropriate prior blocks, $[F_{i-1}, B_i]$ for foward, $[B_{i+1}, F_i]$ for backward) to generate forward and backward blocks, forming a complete chain-of-thought sequence of length $2NL$. At each backward block B_i (particularly at steps $2N, 4N, 6N, \ldots, 2NL$), we extract the flattened weight matrix W^l (rows $2d+1$ to $2d+d^2$ of B_i) from the generated block and measure the squared Frobenius norm of deviation from the ground-truth teacher weights

$$\|W_{\text{CoT}}^l - W_{\text{true}}^l\|_F^2.$$

Result. We repeated the experiment for 5 rounds (each round with a different set of training and validation data) and in each of them we observed that in just 20 epochs of training, the network has successfully learned how to accomplish the

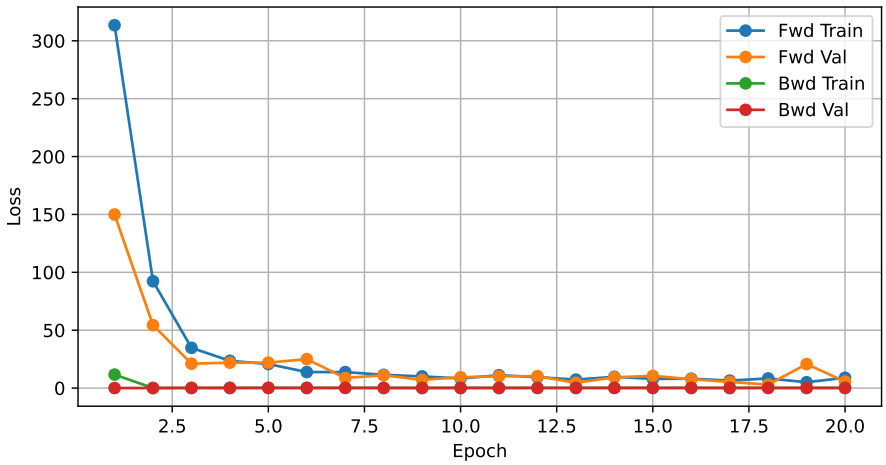

Figure 2. The result of our experiment, four lines denote training loss and validation loss for simulated forward propagation and backward propagation.

forward propagation and the backward propagation. We show the result of our training in Figure 2. The minimal validation losses for forward and backward processes are 3.00 and 0.139.