# Trainable Transformer in Transformer

**Abhishek Panigrahi** [*1] **Sadhika Malladi** [*1] **Mengzhou Xia** [1] **Sanjeev Arora** [1]

## Abstract

Recent works attribute the capability of in-context learning (ICL) in large pre-trained language models to implicitly simulating and fine-tuning an internal model (e.g., linear or 2-layer MLP) during inference. However, such constructions require large memory overhead, which makes simulation of more sophisticated internal models intractable. In this work, we propose a new efficient construction, *Transformer in Transformer* (in short, TINT), that allows a transformer to simulate and fine-tune more complex models during inference (e.g., pre-trained language models). In particular, we introduce innovative approximation techniques that allow a TINT model with less than 2 billion parameters to simulate and fine-tune a 125 million parameter transformer model within a single forward pass. TINT accommodates many common transformer variants and its design ideas also improve the efficiency of past instantiations of simple models inside transformers. We conduct end-to-end experiments to validate the internal fine-tuning procedure of TINT on various language modeling and downstream tasks. For example, even with a limited one-step budget, we observe TINT for a OPT-125M model improves performance by $4-16\%$ absolute on average compared to OPT-125M. These findings suggest that large pre-trained language models are capable of performing intricate subroutines. To facilitate further work, a modular and extensible codebase for TINT is included.

## 1. Introduction

Large transformers (Vaswani et al., 2017) have brought about a revolution in language modeling, with scaling yielding significant advancements in capabilities (Brown et al., 2020; Chowdhery et al., 2022). These capabilities include performing in-context learning or following natural language instructions at inference time.

Researchers have tried to understand how these models can learn new tasks without parameter updates (Garg et al., 2022; von Oswald et al., 2023; Xie et al., 2022; Nanda et al., 2023). A popular hypothesis is that in-context learning corresponds to the transformer (referred to as the *simulator* from now on) simulating gradient-based learning of a smaller model (called *auxiliary* model) that is embedded within it.

From perspective of AI safety and alignment (Amodei et al., 2016; Leike et al., 2018; Askell et al., 2021), the ability of a larger model to use input data (which could be arbitrary in a deployed setting) to implicitly train an auxiliary model feels worrisome. This concern felt minor due to efficiency considerations: previous analyses and experiments required the auxiliary model to be quite tiny compared to the simulator. For instance, simulating and training an auxiliary model that is a linear layer requires tens of millions of parameters in the simulator (Akyurek et al., 2022). This scaling is even more dramatic if the auxiliary model is a multi-layer fully-connected net (Giannou et al., 2023).

Our primary contribution is an explicit and nontrivial construction of a simulator called TINT that explicitly adapts to the context without parameter updates. In particular, we show that a forward pass through a modestly sized TINT can involve gradient-based training of an auxiliary model that is itself a large transformer. For example, we show that TINT with 2B parameters can faithfully simulate fine-tuning a 125M parameter auxiliary transformer in a single forward pass. (Prior constructions would have required trillions of parameters in the simulator for a far simpler auxiliary model.)

Our main result is described in Theorem 1.1, which details how the size of TINT depends on the auxiliary model. Our construction is generally applicable to diverse variants of pre-trained language models. The rest of the paper is structured to highlight the key design choices and considerations in TINT.

1. Section 2 discusses the overall design decisions required to make TINT, including how the simulator can read

[*]Equal contribution [1]Department of Computer Science, Princeton University. Correspondence to: Abhishek Panigrahi <ap34@cs.princeton.edu>, Sadhika Malladi <smalladi@cs.princeton.edu>.

*Proceedings of the $41^{st}$ International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. 

**TINT can efficiently perform simulated gradient descent of an auxiliary model.**

**Theorem 1.1.** *Consider an auxiliary transformer with $L$ layers, $D_{aux}$ embedding dimension, $H_{aux}$ attention heads, and a maximum sequence length of $T_{aux}$. Given a hyperparameter $S$ (see Section 3.1),* TINT *can perform an efficient forward pass (Section 3), compute the simulated gradient (Section 4), and evaluate the updated auxiliary model with a total of*

$$\left(\frac{(c_1S^2+c_3)D_{aux}^2}{\min(H_{aux},S^2)}\cdot D_{aux}^2 + c_2SD_{aux}\min(S^2,H_{aux}) + c_3\frac{T_{aux}D_{aux}S}{\min(H_{aux},S^2)}\right)L$$

*parameters, with constants $c_1, c_2, c_3 < 150$. The* TINT *model has $D_{sim} = SD_{aux}$ embedding dimension and $H_{sim} = \min(S^2, H_{aux})$ attention heads. See Table 3 for a detailed breakdown of the parameters.*

from and write to the auxiliary model and how the data must be formatted.

2. Section 3 uses the linear layer as an example to describe how highly parallelized computation and careful rearrangement of activations enable TINT to efficiently simulate the forward pass of the auxiliary model.

3. Section 4 describes how TINT uses first-order approximations and stop gradients to compute the *simulated gradient* of the auxiliary model.

4. Section 5 performs experiments comparing TINT to suitable baselines in language modeling and in-context learning settings. Our findings validate that the simulated gradient can effectively update large pre-trained auxiliary models. Notably, we instantiate TINT in a highly extensible codebase, making TINT the first such construction to undergo end-to-end evaluation.

Due to the complexity of the construction, we defer the formal details of TINT to the appendix.

**Notations** For a general model, we use $D$ to denote its embedding dimension, $H$ to denote the number of attention heads, $\theta$ to denote its set of all parameters, and $T$ to denote the length of an input sequence. We use subscripts $\cdot_{\text{sim}}$ and $\cdot_{\text{aux}}$ to differentiate the quantities between the TINT and the auxiliary model. For example, $D_{\text{aux}}$ and $D_{\text{sim}}$ refers to the embedding dimension of the auxiliary model and TINT respectively. We use $\boldsymbol{e}_t^{(\ell)} \in \mathbb{R}^{D_{\text{sim}}}$ and $\boldsymbol{x}_t^{(\ell)} \in \mathbb{R}^{D_{\text{aux}}}$ to denote token embeddings in the TINT and the auxiliary model at layer $\ell$ and sequence position $t$ respectively. For a matrix $\boldsymbol{A}$, $\boldsymbol{a}_j$ refers to its $j$th row, and for any vector $\boldsymbol{b}$, $b_j$ refers to its $j$th element.

## 2. Design Considerations

Our goal is to construct a simulator that can train an auxiliary model over the course of an inference pass. This procedure requires four steps:

1. **Forward Pass**: A forward pass to compute the auxiliary model output $f(\xi;\boldsymbol{\theta}_{\text{aux}})$ on training input $\xi$ and a loss $\mathcal{L}$.

2. **Backward Pass**: Backpropagation to compute the gradient of the auxiliary model $\nabla_{\boldsymbol{\theta}_{\text{aux}}}\mathcal{L}(f(\xi;\boldsymbol{\theta}_{\text{aux}}))$.

3. **Parameter Update**: Update the auxiliary model using gradient descent, setting $\boldsymbol{\theta}'_{\text{aux}} = \boldsymbol{\theta}_{\text{aux}} - \eta\nabla_{\boldsymbol{\theta}_{\text{aux}}}\mathcal{L}(f(\xi;\boldsymbol{\theta}_{\text{aux}}))$.

4. **Output**: Output next-token predictions $f(\xi';\boldsymbol{\theta}'_{\text{aux}})$ on a test input $\xi'$ using the updated auxiliary model.

Note that steps 1-3 can be looped to train the auxiliary model for a few steps[1], either on the same training data or on different training data for each step, before evaluating it on the test input (Giannou et al., 2023). The above method highlight two crucial features of the simulator: (1) it has access to some amount of training data, and (2) it can use (i.e., read) and update (i.e., write) the auxiliary model. Below, we discuss how to design a modest-sized simulator around these two considerations.

### 2.1. Input structure

For simplicity, we describe only one update step on a single batch of training data $\xi$ but note that our formal construction and our experiments handle multiple training steps (see Definition 5.1). Steps 1 and 4 show that the simulator must access some training data $\xi$ to train the auxiliary model and some testing data $\xi'$ on which it evaluates the updated auxiliary model. For the sake of illustration we consider the following simple setting: given a sequence of input tokens $\boldsymbol{e}_1, ..., \boldsymbol{e}_T$, we split it into training data $\xi = \boldsymbol{e}_1, ..., \boldsymbol{e}_r$ and testing data $\xi' = \boldsymbol{e}_{r+1}, ..., \boldsymbol{e}_T$.

Suppose $\xi$ contains an in-context input-output exemplar and $\xi'$ contains a test input. Then, the simulator performs a very natural operation of training the auxiliary model on a task-specific example and outputs results for the test example.

On the other hand, if the input is not specially formatted, $\xi$

[1]Looping steps 1-3 scales the depth of the simulator model.

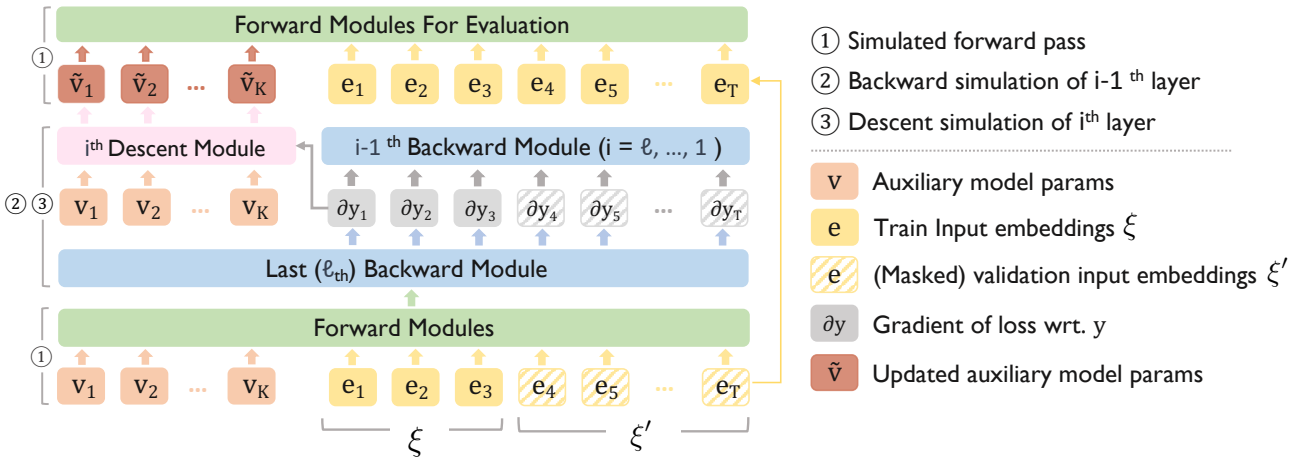


Figure 1: The overall structure of TINT (see Section 2 for an overview). Each forward, backward, and descent module is represented using combinations of linear, self-attention, layernorm, and activation layers. The input consists of prefix embeddings (Definition 2.1) that represent relevant auxiliary model parameters in each layer followed by natural language input. A prefix mask separates the train and test segments of the input (§2.1).

and $\xi'$ may simply contain some natural language tokens. In this case, the simulator is using the first part of the context tokens to do a quick fine-tune of the auxiliary for some task before outputting the subsequent tokens with the auxiliary model. In a worst-case scenario, users might provide harmful contents, leading the model to implicitly fine-tune on them and potentially output even more harmful content.

Our experiments consider many options for splitting a sequence into $\xi$ and $\xi'$, and we defer a more detailed discussion of possible setups to Section 5.

**Accessing Training Labels.** The simulator must be able to see the labels of the training tokens in order to compute the loss $\mathcal{L}$ (usually, the autoregressive cross-entropy loss) in step 1. For example, in Figure 1, when we compute the loss for the token $e_2$ in the second position, we need to use its label $e_3$ in the third position. However, this is not possible if the simulator uses strictly autoregressive attention (Appendix G contains a more general discussion). We thus use a bidirectional attention mask on the training tokens and autoregressive attention on the evaluation portion. We note that encoding relevant (e.g., retrieved) context with bidirectional attention is a popular way to improve autoregressive capabilities in language modeling and natural language tasks (Raffel et al., 2020; Borgeaud et al., 2022; Izacard and Grave, 2020; Izacard et al., 2023; Wang et al., 2023a; Tay et al., 2022). This empirical approach is similar in motivation to how TINT uses a few context tokens to adapt the auxiliary model to a given input. Having established the training and testing data, we can now move to discussing how the simulator can access (i.e., read) and update (i.e., write to) the auxiliary model at inference time.

### 2.2. Read and write access to auxiliary model

As discussed in the start of this section, the simulator must have read and write access to the parameters of the auxiliary model. Crucially, the simulator must do at least two forward passes through the auxiliary model, one with the current parameters $\boldsymbol{\theta}_{\text{aux}}$ and one with the updated parameters $\boldsymbol{\theta}'_{\text{aux}}$. The straightforward way to simulate the forward pass of the auxiliary model would be to store its weights in the simulator's weights and run a forward pass as usual. One can analogously simulate the backward pass according to the loss $\mathcal{L}$ to compute the gradients. However, **the simulator cannot update its own weights at inference time**, so this strategy would not permit the model to write the updated parameters $\boldsymbol{\theta}'_{\text{aux}}$ and later read them when simulating the second forward pass. Therefore, the auxiliary model $\boldsymbol{\theta}_{\text{aux}}$ must be available in the activations of the simulator.

To this end, Wei et al. (2021); Perez et al. (2021) model the simulator after a Turing machine, where the activation $\boldsymbol{e}_t^{(\ell)} \in \mathbb{R}^{D_{\text{sim}}}$ in each layer acts as a workspace for operations, and computation results are copied to and from memory using attention operations. In this paradigm, if $D_{\text{aux}} = 768$, computing a dot product $\langle \boldsymbol{w}, \boldsymbol{x}_t^{(\ell)} \rangle$ with weight $\boldsymbol{w} \in \mathbb{R}^{768}$ requires at least 6.4 million parameters in the simulator[2]. Given the pervasiveness of dot products in neural network modules, this strategy would yield a simulator with trillions of parameters.

[2]Using a feedforward module to mimic the dot product (as in Akyurek et al. (2022), see thm. B.5), where the simulator embedding comprises $[\boldsymbol{w}, \boldsymbol{x}_t] \in \mathbb{R}^{1536}$, necessitates a minimum of 4.7 million parameters. Using an attention module to copy the weight from memory adds another 1.7 million parameters.

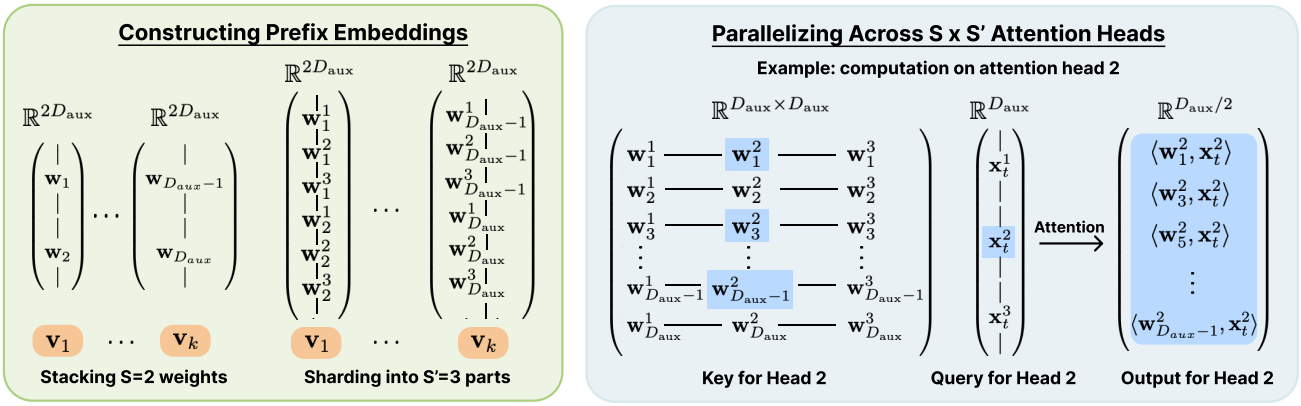


Figure 2: TINT simulates the forward pass of a linear layer with a $H_{\text{sim}}$-head attention layer ($H_{\text{sim}} = 6$ here). We stack $S$ weights per prefix embedding to reduce the number of prefix embeddings required ($S = 2$ here). We furthermore shard each weight and token embedding $\boldsymbol{x}_t$ into $S'$ shards and compute inner products of each shared in parallel using $S \times S'$ attention heads ($S' = 3$ here). Please see Section 3.1.

Alternatively, one can store parameters in the first few context tokens and allow the attention modules to attend to those tokens (Giannou et al., 2023). This removes the need for copying and token-wise operations. Then, the same dot product requires only a self-attention module with 1.7 million parameters. We thus adopt this strategy to provide relevant auxiliary model weights as *prefix embeddings*.

**Definition 2.1** (Prefix Embeddings)**.** $\{\boldsymbol{v}_j^{(\ell)}\}_{j=1}^K$ denotes the $K$ prefix embeddings at the $\ell$th layer in TINT. These contain *relevant* auxiliary model weights or simulated activations.

We now consider how to efficiently simulate the building block of neural networks: matrix-vector multiplication. In the next section, we demonstrate that a careful construction of the prefix embeddings enables efficient parallelizaton of matrix-vector products across attention heads.

## 3. Efficient Forward Propagation

We now discuss how TINT performs a highly efficient forward pass through the auxiliary model. Here, we focus on the linear layer because it is repeated many times in various transformer modules (e.g., in self-attention), so improving the efficiency dramatically reduces TINT's size.

**Definition 3.1** (Linear layer)**.** For a weight $\boldsymbol{W} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$, a linear layer takes $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$.[3]

We compute $\boldsymbol{y}$ coordinate-wise, i.e., $\langle \boldsymbol{w}_i, \boldsymbol{x}_t \rangle$ for all $i \in [D_{\text{aux}}]$, where $\boldsymbol{w}_i$ is the $i$th row of $\boldsymbol{W}$. The simulator represents $\langle \boldsymbol{w}_i, \boldsymbol{x}_t \rangle$ as an attention score between the row $\boldsymbol{w}_i$ and the input $\boldsymbol{x}_t$. So, the input embeddings $\boldsymbol{e}_t$ contain $\boldsymbol{x}_t$ in the first $D_{\text{aux}}$ coordinates, and the rows $\{\boldsymbol{w}_i\}$ of the weight matrix $\boldsymbol{W}$ are in prefix embeddings $\{\boldsymbol{v}_j\}$ (def. 2.1).

We strategically distribute the weights (§3.1) and aggregate the parallelized computation results (§3.2). As we briefly mentioned in the previous section, a straightforward construction of the linear layer would use the context and attention heads inefficiently. Our construction instead parallelizes the computation across attention heads in such a way that aggregating the output of the linear operation can also be conducted efficiently.

### 3.1. Stacking and Sharding

We partition the inner product computation across attention heads by carefully rearranging the weights and activations via stacking and sharding (Figure 2). Instead of representing each weight $\boldsymbol{w}_i$ as its own prefix token $\boldsymbol{v}_i$, we *stack* $S$ weights on top of each other to form each prefix embedding $\boldsymbol{v}_i$. $S$ drives a trade-off between the embedding dimension of the TINT, $D_{\text{sim}} = D_{\text{aux}}S$, and the context length to the TINT, $T_{\text{sim}} = K + T_{\text{aux}}$. We set $S = 4$.

A simple strategy now would be to use different attention heads to operate on different rows; however, this would still use only $S$ attention heads whereas we could parallelize across many more heads. We instead parallelize across more attention heads, where each head is responsible for computing the inner product on a subset of the coordinates. We *shard* each individual weight and the activation into $S'$ parts and compute the inner product on each of the $S'$ parts in parallel We set $S$ and $S'$ such that $H_{\text{sim}} = S \times S'$, thereby using all of TINT heads to efficiently compute the dot products.

[3] Linear layers are applied token-wise, so we can consider a single position $t$ without loss of generality.

### 3.2. Efficient Aggregation

The attention module outputs a sparse matrix with shape $(D_{\text{sim}}/H_{\text{sim}}) \times H_{\text{sim}}$ containing the inner products on various subsets of the coordinates in its entries. To complete the linear forward pass, we need to sum the appropriate terms to form a $D_{\text{sim}}$-length vector with $\boldsymbol{W}\boldsymbol{x}$ in the first $D_{\text{aux}}$ coordinates. Straightforwardly summing along an axis aggregates incorrect terms, since the model was sharded. On the other hand, rearranging the matrix would require an additional $D_{\text{sim}} \times D_{\text{sim}}$ linear layer. Instead, TINT saves a factor of $H_{\text{sim}}\times$ parameters by leveraging the local structure of the attention output. We illustrate this visually in Appendix C.1. This procedure requires $D_{\text{sim}}^2/H_{\text{sim}} + D_{\text{sim}}H_{\text{sim}}$ parameters. This efficient aggregation also compresses the constructions for the TINT's backpropagation modules for layer normalization and activations (Appendices E and F).

## 4. Simulated Gradient

TINT adapts backpropagation to compute gradients (Figure 1). We aim to train a capable (i.e., pre-trained) auxiliary model for just a few steps, so high precision gradients may be unnecessary. Instead, TINT performs an approximate backpropagation. TINT then uses this *simulated gradient* to update the auxiliary model. Prior works computed similar approximate gradients in hopes of more faithfully modeling neurobiology (Scellier and Bengio, 2017; Hinton, 2022) or improving the efficiency of training models (Hu et al., 2021; Malladi et al., 2023). We note that the approximations in the simulated gradients can be made stronger at the cost of enlarging TINT. Indeed, one could construct a simulator to *exactly* perform the procedure outlined in §2, though it would be orders of magnitude larger than TINT. For brevity's sake, we focus on the key approximations and design choices and defer formal details to the appendix.

### 4.1. First-order approximations

We use first-order approximations of gradients to backpropagate through the layer normalization layer.[4] It normalizes the input using its mean and standard deviation across the input dimensions. Since the operation is token-wise, we can consider a single position $t$ without loss of generality.

**Definition 4.1** (Layer normalization)**.** A layer normalization layer $f_{\text{ln}}$ takes input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ and outputs $\boldsymbol{y} = (\boldsymbol{x} - \mu)/\sigma$, where $\mu$ and $\sigma$ denote its mean and standard deviation.

**High precision gradients:** Formally, for input-output pair $(\boldsymbol{x}, \boldsymbol{y})$, we can compute the gradients $\partial_{\boldsymbol{y}}$, $\partial_{\boldsymbol{x}}$ with chain rule:

$$\begin{aligned}\partial_{\boldsymbol{x}} &= \left(\frac{\partial f_{\text{ln}}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^\top \partial_{\boldsymbol{y}} \\ &= \frac{1}{\sigma}\left(\langle \partial_{\boldsymbol{y}}, \boldsymbol{y}\rangle \boldsymbol{y} + \partial_{\boldsymbol{y}} - \frac{1}{D_{\text{aux}}}\sum_{i=1}^{D_{\text{aux}}} \partial_{y_i}\right). \end{aligned} \tag{1}$$

**Inefficiency of exact computation:** A TINT layer simulating backpropagation through an auxiliary's layer normalization layer receives $\partial_{\boldsymbol{y}_t}$ and $\boldsymbol{x}_t$ in its input embeddings. We go through the exact gradient and why it is inefficient.

For exact computation one could first compute $\boldsymbol{y}_t$ using a normalization layer and store in the embeddings. However, inefficiency arises from computing the term $\langle \partial_{\boldsymbol{y}_t}, \boldsymbol{y}_t\rangle \boldsymbol{y}_t$. To calculate $\langle \partial_{\boldsymbol{y}_t}, \boldsymbol{y}_t\rangle \boldsymbol{y}_t$ at each token position $t$, we could either: (1) use a two-layer MLP that focuses on each token separately, or (2) a single self-attention module to treat the operation as a sequence-to-sequence task.

For (1) we could initially compute $\langle \partial_{\boldsymbol{y}_t}, \boldsymbol{y}_t\rangle$ via an MLP, followed by computation of $\langle \partial_{\boldsymbol{y}_t}, \boldsymbol{y}_t\rangle \boldsymbol{y}_t$ using another MLP. The element-wise multiplication in embeddings would be facilitated with a nonlinear activation function like GELU (Akyurek et al., 2022) (refer to thm. B.5 for details). However, this approach would need substantial number of simulator parameters to represent the MLPs.

Alternatively, we could use a single self-attention module. Constructing such a module would require careful engineering to make sure the input tokens only attend to themselves while keeping an attention score of 0 to others. If we used a linear attention, we would need to space out the gradient $\partial_{\boldsymbol{y}_t}$ and $\boldsymbol{x}_t$ in each position $t$, such that the attention score is 0 between different tokens. This would require an embedding dimension proportional to the context length. On the other hand, if we used a softmax attention module, we would need an additional superfluous token in the sequence. Then, a token at position $t$ would attend to itself with attention $\langle \partial \boldsymbol{y}_t, \boldsymbol{y}_t\rangle$ and to the extra token with an attention score of $1 - \langle \partial \boldsymbol{y}_t, \boldsymbol{y}_t\rangle$. The extra token would return a value vector 0. To avoid such inefficiency, we opt for a first-order approximation instead.

**Efficient approximation:** Instead of explicitly computing each term in the chain rule of $\left(\frac{\partial f_{\text{ln}}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^\top \partial_{\boldsymbol{y}}$ in Eq. 1, we instead use a first order Taylor expansion of $f_{\text{ln}}$.

$$f_{\text{ln}}(\boldsymbol{x} + \epsilon\partial_{\boldsymbol{y}}) = f_{\text{ln}}(\boldsymbol{x}) + \epsilon\left(\frac{\partial f_{\text{ln}}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\partial_{\boldsymbol{y}} + \mathcal{O}(\epsilon^2).$$

Rearranging allows us to write

$$\left(\frac{\partial f_{\text{ln}}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\partial_{\boldsymbol{y}} = \frac{1}{\epsilon}\left(f_{\text{ln}}(\boldsymbol{x} + \epsilon\partial_{\boldsymbol{y}}) - f_{\text{ln}}(\boldsymbol{x})\right) + \mathcal{O}(\epsilon).$$

[4] We discuss a layer normalization layer $f_{\text{ln}}$ without scale and bias parameters, but Appendix E contains a general construction.

Similar to the computation of Eq. 1, we can show

$$\frac{\partial f_{\ln}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{1}{\sigma}\left((1 - {D_{\text{aux}}}^{-1})\boldsymbol{I} - f_{\ln}(\boldsymbol{x}) f_{\ln}(\boldsymbol{x})^\top\right).$$

Because $\partial f_{\ln}(\boldsymbol{x})/\partial \boldsymbol{x}$ is symmetric[5], we can write

$$\begin{aligned}\left(\frac{\partial f_{\ln}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^\top \partial_{\boldsymbol{y}} &= \left(\frac{\partial f_{\ln}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\partial_{\boldsymbol{y}} \\ &= \frac{1}{\epsilon}\left(f_{\ln}(\boldsymbol{x} + \epsilon\partial_{\boldsymbol{y}}) - f_{\ln}(\boldsymbol{x})\right) + \mathcal{O}(\epsilon).\end{aligned}$$

Then, ignoring the small error term, we can use just two linear layers, separated by a normalization layer, to simulate the approximation.

### 4.2. Fuzzy backpropagation via stop gradients

Self-attention is inherently quadratic, because it uses the keys and queries to compute attention scores between every possible pair of tokens in the sequence. These scores then linearly combine the value vectors (see def. B.1 for a formal definition). Computing the gradient exactly is thus a very complex operation. Instead, we stop the gradient computation through attention scores in the self-attention layer. For similar reasons, we only update the value parameter in the self-attention module.

**Gradient backpropagation:** For an input, output sequence pair $\{\boldsymbol{x}_t\}, \{\boldsymbol{y}_t\}$, if $\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\}$ denote the intermediate query, key, value vectors, on gradients $\{\partial_{\boldsymbol{y}_t}\}, \{\partial_{\boldsymbol{x}_t}\}$ is given via the chain rule:

$$\partial_{\boldsymbol{x}_t} = \boldsymbol{Q}^\top \partial_{\boldsymbol{q}_t} + \boldsymbol{K}^\top \partial_{\boldsymbol{k}_t} + \boldsymbol{V}^\top \partial_{\boldsymbol{v}_t}. \tag{2}$$

Here, $\boldsymbol{V}, \boldsymbol{K}, \boldsymbol{Q}$ denote the query, key, and value matrices.

**Inefficiency in exact computation:** Here, we demonstrate that simulating computation of the three terms in Eq. 2 is inefficient, because $\partial_{\boldsymbol{q}_t}, \partial_{\boldsymbol{k}_t}$ depend on the derivatives w.r.t. the attention scores. As an example, we focus on $\partial_{\boldsymbol{k}_t}$:

$$\partial_{\boldsymbol{k}_t} = \sum_j a_{t,j}((\partial_{\boldsymbol{y}_t})^\top \boldsymbol{v}_j)(\boldsymbol{k}_j - \sum_{j'} a_{t,j'}\boldsymbol{k}_{j'}).$$

Computing this term would require us at least 2 self-attention layers and an MLP layer. The first attention layer would compute $(\partial_{\boldsymbol{y}_t})^\top \boldsymbol{v}_j$ for different token pairs, similar to the forward simulation of a linear layer with linear attention (§3). These would be then multiplied to the pair-wise attention scores $a_{t,j}$ with an MLP to compute $a_{t,j}((\partial_{\boldsymbol{y}_t})^\top \boldsymbol{v}_j)$, with elementwise product would be facilitated by GeLU non-linearity (thm. B.5). These would be finally used by an attention layer to combine the different key vectors. A similar simulation would be necessary to compute $\partial_{\boldsymbol{q}_t}$.

[5]For a linear function $f$ with matrix $\boldsymbol{W}$, $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \boldsymbol{W}$. Since $\boldsymbol{W}$ may not be a symmetric matrix, this method can't be generally applied to approximately backpropagate linear layers or causal self-attention layers.

Table 1: Language modeling results on WIKITEXT-103. We use $30\%, 50\%, 70\%$ and $90\%$ of sequences for training in the language modeling setting (§5.2). TINT improves the auxiliary model perplexities by $0.3 - 0.7$ absolute on average. The small perplexity difference between the TINT and explicitly updating the auxiliary model suggests that the simulated gradient (Section 4) can still effectively fine-tune the auxiliary model.

| | | Training proportion | | | |
|---|---|---|---|---|---|
| | Evaluating with | 30% | 50% | 70% | 90% |
| GPT-2 | Auxiliary Model | 25.6 | 24.9 | 24.5 | 23.3 |
| | Fine-tuning | 24.9 | 24.0 | 23.5 | 22.2 |
| | TINT | 25.1 | 24.3 | 23.8 | 22.6 |
| OPT-125M | Auxiliary Model | 29.6 | 28.8 | 28.0 | 28.0 |
| | Fine-tuning | 29.0 | 28.2 | 27.4 | 27.4 |
| | TINT | 29.3 | 28.4 | 27.5 | 27.4 |

**Stop gradients through query and key vectors:** In order to reduce the necessary resources, we ignore the query and key gradients in Eq. 2. When we ignore these gradient components, $\{\partial_{\boldsymbol{x}_t}\}$ can be simplified as

$$\partial_{\boldsymbol{x}_t} \approx \boldsymbol{V}^\top \partial_{\boldsymbol{v}_t} = \boldsymbol{V}^\top \sum_j a_{j,t}\partial_{\boldsymbol{y}_t}. \tag{3}$$

A single self-attention layer can compute this by using the attention scores to combine the token-wise gradients.

**Why won't it hurt performance?** Estimating $\partial_{\boldsymbol{x}_t}$ as described is motivated by recent work (Malladi et al., 2023) showing that fuzzy gradient estimates don't adversely affect fine-tuning of pre-trained models. Furthermore, we theoretically show that when the attention head for each position pays a lot of attention to a single token (i.e., behaves like hard attention (Perez et al., 2021)), the approximate gradient in Eq. 3 is entry-wise close to the true gradients (thm. D.5).

The other approximation is to update only the value parameters $\boldsymbol{V}$ of the auxiliary model (§D). This is motivated by parameter efficient fine-tuning methods like LORA (Hu et al., 2021) and IA3 (Liu et al., 2022), which restrict the expressivity of the gradient updates without degrading the quality of the resulting model. We similarly show in the next section that the simulated gradients in TINT can effectively tune large pre-trained transformers.

## 5. Experiments

We evaluate the performance of the TINTs constructed using GPT2 and OPT-125M as auxiliary models. The findings from our experiments in the language modeling and

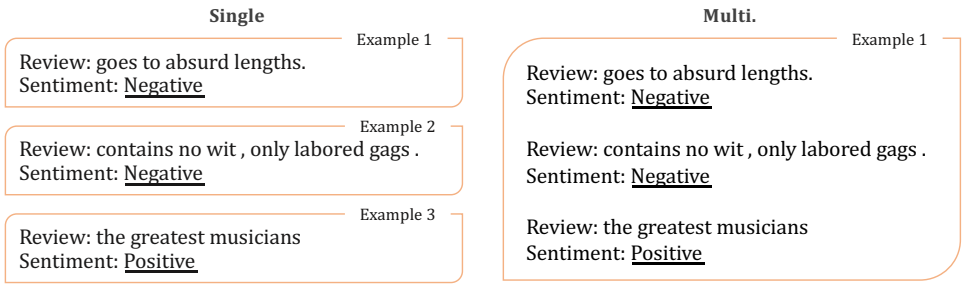


Figure 3: Different settings in few-shot learning ($k = 3$) using TINT. The **Single** mode (left) treats each example as a training datapoint, and the auxiliary model is updated with a batch of inputs (see def. 5.1). The **Multi.** mode (right) concatenates all examples to form a single input and uses batch size 1 in def. 5.1. For **Label loss**, only underlined label words are used as training signal, while **full context loss** includes all tokens.

in-context learning settings confirm that fine-tuning with the simulated gradients (Section 4) still allows for effective learning in the auxiliary model. We loop the training steps (i.e., steps 1-3) outlined in Section 2 to accommodate solving real-world natural language tasks. We formalize the setting below.

### 5.1. Setting: $N$-step Fine-Tuning

We formalize the procedure in Section 2 to construct a suitable setting in which we can compare TINT to explicitly training the auxiliary model.

**Definition 5.1** ($N$-step Fine-Tuning)**.** Given a batch of training datapoints $\xi_1, \cdots, \xi_B$ and a validation input $\xi'$, we compute and apply gradient updates on the auxiliary model $\boldsymbol{\theta}_{\text{aux}}$ for timesteps $t = 0, ..., N-1$ as

$$\boldsymbol{\theta}_{\text{aux}}^{t+1} = \boldsymbol{\theta}_{\text{aux}}^{t} - \eta \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\xi_i; \boldsymbol{\theta}_{\text{aux}}^{t}))$$

where $\eta$ is the learning rate and $\mathcal{L}$ is a self-supervised loss function on each input $\xi_i$. Then, we evaluate the model $\boldsymbol{\theta}_{\text{aux}}^{N}$ on $\xi'$. $\boldsymbol{\theta}_{\text{aux}}^{0}$ denotes the pre-trained auxiliary model.

Below, we instantiate this setting with text inputs of different formats and different self-supervised loss functions $\mathcal{L}$. To manage computational demands, we limit $N$ to 3 or fewer.[6]

### 5.2. Case Study: Language Modeling

The first case we consider is language modeling, where the input data $\boldsymbol{e}_1, ..., \boldsymbol{e}_T$ is natural language without any additional formatting. We use a batch size of 1 in def. 5.1, and delegate $\xi_1 = \boldsymbol{e}_1, ..., \boldsymbol{e}_t$ and $\xi' = \boldsymbol{e}_{t+1}, ..., \boldsymbol{e}_T$. The loss $\mathcal{L}$ is the sum of the token-wise autoregressive cross-entropy loss in the sequence $\xi_1$. For example, given an input Machine learning is a useful tool for solving problems., we use the red part as the training data $\xi_1$, and the brown part as the validation data $\xi'$. We perform language modeling experiments on WIKITEXT-103 (Merity et al., 2016) and vary the number of tokens $t$ used as training data $\xi$.

**Results.** In Table 1, we observe that TINT achieves a performance comparable to explicit fine-tuning of the auxiliary model, indicating that the simulated gradient (Section 4) is largely effective for fine-tuning. Both TINT and explicitly fine-tuning the auxiliary model show improvement over the base model, confirming that minimal tuning on the context indeed enhances predictions on the test portion.

### 5.3. Case Study: In-Context Learning

For in-context learning, we consider input data to be a supervised classification task transformed into a next-token prediction task using surrogate labels (see Figure 3). Using binary sentiment classification of movie reviews as an example, given an input (e.g., the review), the model's predicted label is computed as follows. First, we design a simple task-specific prompt (e.g., "Sentiment:") and select label words $c_1, ..., c_n$ to serve as surrogates for each class (e.g., "positive" and "negative"). Then, we provide the input along with the prompt to the model, and the label assigned the highest probability is treated as the model's prediction. We describe the zero-shot and few-shot settings below.

**Zero-shot.** In the zero-shot setting, we are given text with the first $T-1$ tokens as the input text and final token as the surrogate text label. Hence, we adapt def. 5.1 to use batch size $B = 1$, training data $\xi_1 = x_1, ..., x_{T-1}$, and testing data $\xi' = x_T$. The loss $\mathcal{L}$ is again the sum of the token-wise autoregressive cross-entropy losses.

**Few-shot.** In the few-shot setting, we are given input texts that are a concatenation of $k$ sequences $\xi_1, \cdots, \xi_k$. Each sequence contains the input text followed by the surrogate label for the in-context exemplar. These $k$ exemplars are followed by test data $\xi'$. In this case, we can compute the

[6] Performing many gradient steps scales the depth of TINT and makes experimentation computationally infeasible.

Table 2: Zero-shot and few-shot in-context learning results across 7 downstream tasks. All the few-shot results are averaged over three training seeds. TINT consistently surpasses its auxiliary model and achieves comparable performance to Fine-tuninguation. TINT outperforms auxiliary models by $3-4\%$ and $12-16\%$ absolute points on average in 0-shot and 32-shot experiments respectively. TINT performs competitively with a similar-sized pre-trained model (OPT-1.3B) in both 0-shot and 32-shot settings. We show the standard deviation for few-shot settings in parentheses.

| Model | Shots | Subj | AGNews | SST2 | CR | MR | MPQA | Amazon | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | ***Without Calibration*** | | | | |
| OPT-125M | 0 | 64.0 | 66.0 | 70.5 | 64.5 | 71.0 | 68.0 | 76.5 | 68.6 |
| OPT-1.3B | 0 | 59.0 | 55.5 | 54.0 | 50.5 | 52.5 | 74.0 | 57.0 | 57.5 |
| OPT-125M Fine-tuning | 0 | 71.0 | 67.0 | 79.5 | 71.5 | 70.0 | 68.0 | 85.5 | 73.2 |
| OPT-125M TINT | 0 | 67.5 | 66.0 | 76.5 | 69.0 | 76.0 | 70.5 | 78.5 | 72.0 |
| OPT-125M | 32 | $58.7_{(4.9)}$ | $33.7_{(8.4)}$ | $50.8_{(1.2)}$ | $51.3_{(1.9)}$ | $50.0_{(0.0)}$ | $54.3_{(2.5)}$ | $55.0_{(6.7)}$ | $50.5_{(1.9)}$ |
| OPT-1.3B | 32 | $74.2_{(6.1)}$ | $71.3_{(5.3)}$ | $89.8_{(3.6)}$ | $71.5_{(4.5)}$ | $68.3_{(6.1)}$ | $81.7_{(3.3)}$ | $70.3_{(9.9)}$ | $75.3_{(0.4)}$ |
| OPT-125M Fine-tuning | 32 | $78.0_{(1.4)}$ | $66.7_{(1.6)}$ | $71.5_{(1.4)}$ | $73.7_{(3.3)}$ | $72.0_{(0.0)}$ | $80.7_{(0.6)}$ | $79.8_{(0.2)}$ | $74.6_{(2.7)}$ |
| OPT-125M TINT | 32 | $82.3_{(2.7)}$ | $69.3_{(0.9)}$ | $73.7_{(0.8)}$ | $75.7_{(1.9)}$ | $72.3_{(1.2)}$ | $83.2_{(1.0)}$ | $78.2_{(0.2)}$ | $76.4_{(0.7)}$ |
| | | | | | ***With Calibration*** | | | | |
| OPT-125M | 0 | 64.0 | 66.0 | 53.0 | 54.5 | 52.5 | 55.5 | 58.0 | 57.6 |
| OPT-1.3B | 0 | 73.5 | 61.5 | 57.5 | 53.0 | 54.5 | 79.5 | 61.0 | 62.9 |
| OPT-125M Fine-tuning | 0 | 62.5 | 66.0 | 60.5 | 53.5 | 54.0 | 56.5 | 74.5 | 61.1 |
| OPT-125M TINT | 0 | 64.0 | 66.0 | 56.5 | 59.0 | 53.5 | 62.0 | 66.5 | 61.1 |
| OPT-125M | 32 | $83.5_{(2.4)}$ | $40.7_{(10.4)}$ | $50.8_{(0.8)}$ | $67.7_{(4.1)}$ | $57.7_{(10.8)}$ | $79.2_{(8.4)}$ | $56.0_{(8.1)}$ | $62.2_{(2.7)}$ |
| OPT-1.3B | 32 | $51.8_{(1.9)}$ | $66.2_{(3.1)}$ | $93.7_{(1.0)}$ | $82.8_{(2.8)}$ | $91.3_{(1.9)}$ | $83.5_{(2.5)}$ | $92.0_{(2.9)}$ | $80.2_{(0.7)}$ |
| OPT-125M Fine-tuning | 32 | $87.2_{(0.2)}$ | $67.2_{(0.6)}$ | $72.8_{(5.9)}$ | $73.3_{(2.6)}$ | $66.7_{(7.4)}$ | $81.5_{(3.7)}$ | $70.3_{(2.1)}$ | $74.1_{(2.9)}$ |
| OPT-125M TINT | 32 | $85.3_{(1.9)}$ | $67.3_{(0.6)}$ | $71.8_{(3.8)}$ | $70.7_{(1.9)}$ | $63.7_{(0.2)}$ | $83.5_{(1.6)}$ | $77.5_{(1.2)}$ | $74.3_{(1.4)}$ |

gradient updates to $\boldsymbol{\theta}_{\text{aux}}$ in two different ways (Figure 3). The first setting, denoted **Single**, treats the $k$ sequences as a batch of $B = k$ training datapoints $\xi_1, ..., \xi_B$. The second setting, denoted **Multi**, treats the concatenation of the $B$ sequences as a single training datapoint $\xi_1$. Furthermore, $\mathcal{L}$ for a training datapoint can be defined in two different ways. The first setting, denoted as **Full context loss**, defines $\mathcal{L}$ for a training datapoint $\xi_i$ as the sum of cross entropy loss over all tokens. The second setting, denoted as **Label loss**, defines $\mathcal{L}$ for a training datapoint $\xi_i$ in def. 5.1 as the sum of cross entropy loss over the surrogate label tokens.

**Tasks.** We evaluate 7 classification tasks for zero-shot and few-shot settings: SST-2 (Socher et al., 2013), MR (Pang and Lee, 2004), CR (Hu and Liu, 2004), MPQA (Wiebe et al., 2005), Amazon Polarity (Zhang et al., 2015), AGNews (Zhang et al., 2015), and Subj (Pang and Lee, 2005).

**Model.** We compare a TINT model that uses an OPT-125M pre-trained model as its auxiliary model against two alternative approaches: (1) directly fine-tuning OPT-125m, and (2) performing standard evaluation using OPT-1.3b, which is of a similar size to TINT.[7]

[7]Our construction is generally applicable to diverse variants of pre-trained language models (Appendix J).

**Calibration:** We report the performance in Table 2 in two settings: no calibration, and with calibration. If using calibration, the predicted probabilities of the surrogate labels are normalized using just the prompt as input.

$$\text{No Calibration: } \underset{c_i}{\operatorname{argmax}} \Pr[c_i \mid \text{input, prompt}]$$

$$\text{Calibration: } \arg\max_{c_i} \frac{\Pr[c_i \mid \text{input, prompt}]}{\Pr[c_i \mid \text{prompt}]}$$

This is a widely used calibration technique (Holtzman et al., 2021) for prompting language models.

**Observations.** We observe that inferences passes through TINT perform on par with directly fine-tuning the auxiliary model, affirming the validity of the construction design (see Section 2). As expected, TINT outperforms the base auxiliary model, since it simulates training the auxiliary model. More intriguingly, TINT demonstrates performance comparable to a pre-trained model of similar size (OPT-1.3B). This suggests that the capabilities of existing pre-trained models may be understood via the simulation of smaller auxiliary models. We observe that calibration may not always be beneficial in every setting.[8] However, even with calibration, TINT remains competitive to fine-tuning

[8]Such inconsistencies in the calibration method have been observed in previous works (Brown et al., 2020).

of OPT models. The performance of OPT-1.3B improves with calibration. In this case, TINT lags behind OPT-1.3B in the few-shot setting. For further details and results of the experiments, please refer to Appendix K.

## 6. Related Work

**Gradient-based learning and in-context learning:** Several works relate in-context learning to gradient-based learning algorithms. Bai et al. (2023) explicitly constructed transformers to simulate simple gradient-based learning algorithms. Mahankali et al. (2023); Ahn et al. (2023) suggested one attention layer mimics gradient descent on a linear layer, and Zhang et al. (2023b) showed polynomial convergence. Cheng et al. (2023); Han et al. (2023) extended these ideas to non-linear attentions. Experiments in Dai et al. (2022) suggest that LLM activations during in-context learning mirror fine-tuned models. These works focus on using a standard transformer for the simulator and hence cannot accommodate more complex auxiliary models; on the other hand, our work uses structural modifications and approximations to construct an efficient simulator for complex auxiliary models. Our work in contrast attempts to build even stronger transformers by introducing few structural modifications that can run gradient descent on auxiliary transformers.

**Transformer Expressivity:** Perez et al. (2021); Pérez et al. (2019) show that Transformers with hard attention are Turing complete, and Wei et al. (2021) construct transformers to study statistical learnability, but the proposed constructions are extremely large. Other works have investigated encoding specific algorithms in smaller simulators, e.g. bounded-depth Dyck languages (Yao et al., 2021), modular prefix sums (Anil et al., 2022), adders (Nanda et al., 2023), regular languages (Bhattamishra et al., 2020), and sparse logical predicates (Edelman et al., 2022). Liu et al. (2023) aim to understand automata-like mechanisms within transformers. Ba et al. (2016) connect self-attention and fast weight programmers (FWPs), which compute input-dependent weight updates during inference. Follow-up works (Schlag et al., 2021; Irie et al., 2021) use self-attention layers to update linear and recurrent networks during inference. Clark et al. (2022) add and efficiently tune Fast Weights Layers (FWL) on a frozen pre-trained model.

## 7. Discussion

We present a parameter-efficient construction TINT capable of simulating gradient descent on an internal transformer model during inference. Using fewer than 2 billion parameters, it can simulate fine-tuning a 125 million transformer (e.g., GPT-2) internally, dramatically reducing the scale required by previous works. Language modeling and in-context learning experiments demonstrate that the efficient approximations still allow the TINT to fine-tune the model. Our work emphasizes that the inference behavior of complex models may rely on the training dynamics of smaller models. As such, the existence of TINT has strong implications for interpretability and AI alignment research.

Similar to prior research in this area, our insights into existing pre-trained models are limited. TINT was designed to understand the power of in-context reasoning with an explicit construction, and thereby, understand the safety risk of transformers trained with moderate compute. Hence, we introduce two major architectural modifications. TINT uses bidirectional attention for efficiently computing the loss on training portion $\xi$ (§2). Furthermore, we use prefix embeddings to efficiently represent the relevant auxiliary model parameters in each layer of TINT (§2). Both of these design principles are largely motivated by existing perfomant architectures (Tay et al., 2022; Raffel et al., 2020; Cheng et al., 2023; Izacard et al., 2023; Borgeaud et al., 2022) and fine-tuning strategies (Liu et al., 2021; Lester et al., 2021; Zhang et al., 2023a; Li and Liang, 2021). However, because of the architecture modifications, TINT cannot be used to explain the in-context capability in existing popular autoregressive models (Touvron et al., 2023; Brown et al., 2020).

TINT provides a possible connection between fine-tuning and in-context reasoning with transformer models. As such, inference time behaviors of large language models may require understanding the training dynamics of smaller transformers. On the other hand, such a connection can also lead to explorations on improved architecture designs by measuring generalization behaviors of the underlying simulated auxiliary model (Li and Zhang, 2021; Ju et al., 2022). Furthermore, we have not yet examined potential biases that may arise in the auxiliary models due to one-step gradient descent. We plan to investigate these aspects in future work.

## Impact Statement

We note that the construction of TINT does not appear to increase the probability of harmful behavior, because the construction's primary objective is to implicitly tune an internal model (§2). Such tuning has been possible for a long time and is not made more expressive by TINT.

Our findings suggest that existing transformer-based language models can plausibly possess the ability to learn and adapt to context by internally fine-tuning a complex model *even during inference*. Consequently, although users are unable to directly modify deployed models, these models may still undergo dynamic updates while processing a context left-to-right, resulting in previously unseen behavior by the time the model reaches the end of the context. This has significant implications for the field of model alignment. It is challenging to impose restrictions on a model that can perform such dynamics updates internally, so malicious content can influence the output of deployed models.

Alternatively, we recognize the potential benefits of pre-training constructed models that integrate explicit fine-tuning mechanisms. By embedding the functionalities typically achieved through explicit fine-tuning, such as detecting malicious content and intent within the models themselves, the need for external modules can be mitigated. Pre-training the constructed model may offer a self-contained solution for ensuring safe and responsible language processing without relying on external dependencies.

## Acknowledgements

The authors acknowledge funding from NSF, ONR, Simons Foundation, and DARPA. We thank Danqi Chen, Jason Lee, Zhiyuan Li, Kaifeng Lyu, Simran Kaur, Tianyu Gao, and Colin Wang for their suggestions and helpful discussions at different stages of our work. We thank the anonymous reviewers and the Area Chairs of NeurIPS'23, ICLR'24, and ICML'24 assigned to our paper for their helpful and detailed reviews and meta-reviews to improve the quality of our paper.

## References

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *arXiv preprint arXiv:2306.00297*, 2023.

Ekin Akyurek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *arXiv preprint arXiv:2207.04901*, 2022.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *arXiv preprint arXiv:2306.04637*, 2023.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. *arXiv preprint arXiv:2302.03025*, 2023.

Kevin Clark, Kelvin Guu, Ming-Wei Chang, Panupong Pasupat, Geoffrey Hinton, and Mohammad Norouzi. Meta-learning fast weight language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9751–9757, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.661.

Arthur Conmy, Augustine N Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*, 2023.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models secretly perform gradient descent as meta-optimizers, 2022.

Benjamin L Edelman, Surbhi Goel, Sham Kakade, and Cyril Zhang. Inductive biases and variable creation in self-attention mechanisms. In *International Conference on Machine Learning*, pages 5793–5831. PMLR, 2022.

N Elhage, N Nanda, C Olsson, T Henighan, N Joseph, B Mann, A Askell, Y Bai, A Chen, T Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

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

Angeliki Giannou, Shashank Rajput, Jy yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers, 2023.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pages 2337–2346. PMLR, 2019.

Michael Hahn and Navin Goyal. A theory of emergent in-context learning as implicit structure induction. *arXiv preprint arXiv:2303.07971*, 2023.

Chi Han, Ziqi Wang, Han Zhao, and Heng Ji. In-context learning of large language models explained as kernel regression. *arXiv preprint arXiv:2305.12766*, 2023.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

Ari Holtzman, Peter West, Vered Shwartz, Yejin Choi, and Luke Zettlemoyer. Surface form competition: Why the highest probability answer isn't always right. *arXiv preprint arXiv:2104.08315*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177, 2004.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ot2ORiBqTa1.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251): 1–43, 2023. URL http://jmlr.org/papers/v24/23-0037.html.

Hui Jiang. A latent space theory for emergent abilities in large language models. *arXiv preprint arXiv:2304.09960*, 2023.

Haotian Ju, Dongyue Li, and Hongyang R Zhang. Robust fine-tuning of deep neural networks with hessian-based generalization guarantees. In *International Conference on Machine Learning*, pages 10431–10461. PMLR, 2022.

Ananya Kumar, Ruoqi Shen, Sébastien Bubeck, and Suriya Gunasekar. How to fine-tune vision models with sgd, 2022.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Dongyue Li and Hongyang Zhang. Improved regularization and robustness for fine-tuning in neural networks. *Advances in Neural Information Processing Systems*, 34: 27249–27262, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*, 2023.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965, 2022.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

Arvind Mahankali, Tatsunori B Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv preprint arXiv:2307.03576*, 2023.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Vota6rFhBQ.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 271–278, 2004.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, 2005.

Jorge Perez, Pablo Barcelo, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021. URL http://jmlr.org/papers/v22/20-302.html.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGBdo0qFm.

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

Sashank J Reddi, Sobhan Miryoosefi, Stefani Karp, Shankar Krishnan, Satyen Kale, Seungyeon Kim, and Sanjiv Kumar. Efficient training of language models using few-shot learning. 2023.

Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. A mathematical exploration of why language models help solve downstream tasks. *arXiv preprint arXiv:2010.03648*, 2020.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight memory systems. *CoRR*, abs/2102.11174, 2021. URL https://arxiv.org/abs/2102.11174.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, et al. Ul2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Johannes von Oswald, Eyvind Niklasson, Maximilian Schlegel, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, Razvan Pascanu, et al. Uncovering mesa-optimization algorithms in transformers. *arXiv preprint arXiv:2309.05858*, 2023.

Boxin Wang, Wei Ping, Peng Xu, Lawrence McAfee, Zihan Liu, Mohammad Shoeybi, Yi Dong, Oleksii Kuchaiev, Bo Li, Chaowei Xiao, Anima Anandkumar, and Bryan Catanzaro. Shall we pretrain autoregressive language models with retrieval? a comprehensive study. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7763–7786, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.482. URL https://aclanthology.org/2023.emnlp-main.482.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *NeurIPS ML Safety Workshop*, 2022. URL https://openreview.net/forum?id=rvi3Wa768B-.

Xinyi Wang, Wanrong Zhu, and William Yang Wang. Large language models are implicitly topic models: Explaining and finding good demonstrations for in-context learning. *arXiv preprint arXiv:2301.11916*, 2023b.

Colin Wei, Yining Chen, and Tengyu Ma. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *CoRR*, abs/2107.13163, 2021. URL https://arxiv.org/abs/2107.13163.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR, 2021.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39:165–210, 2005.

Noam Wies, Yoav Levine, and Amnon Shashua. The learnability of in-context learning. *arXiv preprint arXiv:2303.07895*, 2023.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=RdJVFCHjUMI.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. *arXiv preprint arXiv:2105.11115*, 2021.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023a.

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

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Yufeng Zhang, Fengzhuo Zhang, Zhuoran Yang, and Zhaoran Wang. What and how does in-context learning learn? bayesian model averaging, parameterization, and generalization. *arXiv preprint arXiv:2305.19420*, 2023c.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

## Contents

## Brief overview of the appendix

In Appendix A, we report few additional related works. In Appendix B, we present all the important notations used to present the design of TINT. In Appendices C to F, we present the simulation details of all operations on linear, self-attention, layer normalization, and activation layers respectively for an auxiliary model. In Appendix G, we present the details for simulating loss computation with the language model head of the auxiliary model. In Appendix I, we discuss simulation of additional modules necessary to simulate transformer variants like LLaMA (Touvron et al., 2023) and BLOOM (Scao et al., 2022). Finally, in Appendix K, we discuss the deferred experimental details from the main paper.

## A. Additional related works

**Interpretability:** Mechanistic interpretability works reverse-engineer the algorithms simulated by these models (Elhage et al., 2021; Olsson et al., 2022; Wang et al., 2022; Nanda et al., 2023; Chughtai et al., 2023; Conmy et al., 2023). These works study local patterns, e.g. activations and attention heads, to derive interpretable insights. Other works (Weiss et al., 2021; Lindner et al., 2023) use declarative programs to algorithmically describe transformer models. Zhou et al. (2023) use these to explain task-specific length generalization of transformer models.

**Alternative Explanations for ICL:** Some works study ICL using a Bayesian framework. Xie et al. (2022) model pretraining data as a mixture of HMMs and cast ICL identifying one such component. Hahn and Goyal (2023) later modeled language as a compositional grammar, and propose ICL as a composition of operations. (Zhang et al., 2023c; Jiang, 2023; Wang et al., 2023b; Wies et al., 2023) further strengthen this hypothesis by generalizing the underlying latent space. On the other hand, careful experiments in Chan et al. (2022) show that data distributional properties (e.g. Zipf's law) drive in-context learning in transformers.

**Transfer learning:** Our construction uses a pre-trained model to initialize a larger transformer, which is similar to several other more empirically oriented works (Gong et al., 2019; Reddi et al., 2023).

## B. Notations

For simplicity, we repeat notations from the main paper. Let $D$ denote the embedding dimension for a token and $T$ denote the length of an input sequence. $H$ denotes the number of attention heads. With the exception of contextual embeddings, we use subscripts to indicate if the quantity is from TINT or from the auxiliary model. For example, $D_{\text{aux}}$ refers to the embedding dimension and $D_{\text{sim}}$ refers to the TINT embedding dimension. For contextual embeddings, we use $\boldsymbol{e}_t^{(\ell)} \in \mathbb{R}^{D_{\text{sim}}}$ to denote activations in TINT and $\boldsymbol{x}_t^{(\ell)} \in \mathbb{R}^{D_{\text{aux}}}$ to denote activations in the auxiliary model, where $\ell$ is the layer and $t$ is the sequence position. When convenient, we drop the superscript that represents the layer index and the subscript that represents the position index. For a matrix $\boldsymbol{A}$, $\boldsymbol{a}_j$ refers to its $j$th row, and for any vector $\boldsymbol{b}$, $b_j$ refers to its $j$th element. However, at a few places, for typographical reasons, for a matrix $\boldsymbol{A}$, we have also used $(\boldsymbol{A})_j$ to refer to its $j$th row, and for any vector $\boldsymbol{b}$, $(\boldsymbol{b})_j$ to refer to its $j$th element. TINT uses one-hot positional embeddings $\{\boldsymbol{p}_i^{\text{TINT}} \in \mathbb{R}^{T_{\text{sim}}}\}_{i \le T_{\text{sim}}}$.

Table 3: Number of parameters of TINT for the forward, backward, and gradient update operations on various modules. For simplicity, we have ignored biases in the following computation. We set $S = 4$, i.e. stack 4 weights in each prefix embedding. We set $H_{\text{sim}} = 12$ for OPT-125M and $H_{\text{sim}} = 16$ for the other models, $D_{\text{sim}} = 4D_{\text{aux}}$ for all the models, and $T_{\text{sim}} = T_{\text{aux}} + K$, with $T_{\text{aux}} = 2048$ for OPT models, and $K = D_{\text{aux}}/4$. $Q = 4Q_{split} + 3T_{\text{sim}}D_{\text{sim}}/H_{\text{sim}}$, where $Q_{split} = \frac{1}{H_{\text{sim}}}(D_{\text{sim}})^2 + H_{\text{sim}}D_{\text{sim}}$, denotes the number of parameters in a TINT Linear Forward module (Section 3).

| | Module Size | | | |
|---|---|---|---|---|
| Module Name | Forward | Backward | Descent | Total |
| Linear layer | $Q$ | $Q$ | $Q$ | $3Q$ |
| Layer norms | $Q$ | $Q + 2D_{\text{sim}}H_{\text{sim}}$ | $Q$ | $3Q + 2D_{\text{sim}}H_{\text{sim}}$ |
| Self-Attention | $2Q$ | $2Q$ | $2Q$ | $6Q$ |
| Activation | $Q_{split}$ | $2D_{\text{sim}}H_{\text{sim}}$ | 0 | $Q_{split} + 2D_{\text{sim}}H_{\text{sim}}$ |
| Self-Attention block | $4Q$ | $4Q + 2D_{\text{sim}}H_{\text{sim}}$ | $4Q$ | $12Q + 2D_{\text{sim}}H_{\text{sim}}$ |
| Feed-forward block | $3Q + Q_{split}$ | $3Q + 4D_{\text{sim}}H_{\text{sim}}$ | $3Q$ | $9Q + 4D_{\text{sim}}H_{\text{sim}}$ |
| Transformer block | $7Q + Q_{split}$ | $7Q + 6D_{\text{sim}}H_{\text{sim}}$ | $7Q$ | $21Q + 6D_{\text{sim}}H_{\text{sim}} + Q_{split}$ |
| Transformer | $7QL + LQ_{split}$ | $(7Q + 6D_{\text{sim}}H_{\text{sim}})L$ | $7QL$ | $(21Q + 6D_{\text{sim}}H_{\text{sim}} + Q_{split})L$ |
| OPT-125M | 0.4B | 0.4B | 0.4B | 1.2B |
| OPT-350M | 1.2B | 1.1B | 1.1B | 3.4B |
| OPT-1.3B | 3.7B | 3.6B | 3.5B | 10.8B |
| OPT-2.7B | 7.4B | 7.2B | 7.2B | 21.8B |

We differentiate the parameters of the auxiliary model and TINT by using an explicit superscript TINT for TINT parameters, for example, the weights of a linear layer in TINT will be represented by $\boldsymbol{W}^{\text{TINT}}$. We use two operations throughout: $\text{SPLIT}_h$ and VECTORIZE. Function $\text{SPLIT}_h : \mathbb{R}^d \to \mathbb{R}^{h \times \lfloor d/h \rfloor}$ takes an input $\boldsymbol{x} \in \mathbb{R}^d$ and outputs $H$ equal splits of $\boldsymbol{x}$, for any arbitrary dimension $d$. Function $\text{VECTORIZE} : \mathbb{R}^{h \times d} \to \mathbb{R}^{dh}$ concatenates the elements of a sequence $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i \le h}$ into one single vector, for any arbitrary $d$ and $h$.

**Auxiliary model's self-attention** We first start with the definition of a single head self-attention layer for the auxiliary model. The definition can be easily extended to a multi-head self-attention layer. A self-attention layer first computes the query, key, and value vectors at each position by token-wise linear transformations of the input embeddings. The query and the key vectors are used to compute pairwise self-attention scores. These scores are then used to linearly combine the value vectors.

**Definition B.1** (Auxiliary model softmax self-attention)**.** A self-attention layer with parameters $\{\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V\}$ takes a sequence $\{\boldsymbol{x}_t\}_{t \le T_{\text{aux}}}$ and outputs a sequence $\{\boldsymbol{y}_t\}_{t \le T_{\text{aux}}}$, such that

$$\boldsymbol{y}_t = \sum_j a_{t,j}\boldsymbol{v}_j, \qquad \text{with } a_{t,j} = \text{softmax}(\boldsymbol{K}\boldsymbol{q}_t)_j, \quad \boldsymbol{q}_t = \boldsymbol{W}_Q\boldsymbol{x}_t, \quad \boldsymbol{k}_t = \boldsymbol{W}_K\boldsymbol{x}_t, \quad \boldsymbol{v}_t = \boldsymbol{W}_V\boldsymbol{x}_t,$$

for all $t \le T_{\text{aux}}$, and $\boldsymbol{K} \in \mathbb{R}^{T_{\text{aux}} \times D_{\text{aux}}}$ defined with rows $\{\boldsymbol{k}_t\}_{t=1}^{T_{\text{aux}}}$.

**TINT Attention Module** We modify the usual attention module to include the position embeddings $\{\boldsymbol{p}_i^{\text{TINT}} \in \mathbb{R}^{T_{\text{sim}}}\}_{i \le T_{\text{sim}}}$. In usual self-attention modules, the query, key, and value vectors at each position are computed by token-wise linear transformations of the input embeddings. In TINT's Attention Module, we perform additional linear transformations on the position embeddings, using parameters $\boldsymbol{W}_Q^p, \boldsymbol{W}_K^p, \boldsymbol{W}_V^p$, and decision vectors $\lambda^Q, \lambda^K, \lambda^V \in \mathbb{R}^{H_{\text{sim}}}$ decide whether to add these transformed position vectors to the query, key, and value vectors of different attention heads. For the following definition, we use $\widehat{e}$ to represent input sequence and $\widetilde{e}$ to represent the output sequence: we introduce these general notations below to avoid confusion with the notations for token and prefix embeddings for TINT illustrated in Figure 1.

**Definition B.2** (TINT's self-attention with $H_{\text{sim}}$ heads)**.** For parameters $\{\boldsymbol{W}_Q^{\text{TINT}}, \boldsymbol{W}_K^{\text{TINT}}, \boldsymbol{W}_V^{\text{TINT}} \in \mathbb{R}^{D_{\text{sim}} \times D_{\text{sim}}}\}$, $\{\boldsymbol{b}_Q^{\text{TINT}}, \boldsymbol{b}_K^{\text{TINT}}, \boldsymbol{b}_V^{\text{TINT}} \in \mathbb{R}^{D_{\text{sim}}}\}$, $\{\boldsymbol{W}_Q^p, \boldsymbol{W}_K^p, \boldsymbol{W}_V^p \in \mathbb{R}^{T_{\text{sim}} \times D_{\text{sim}}/H_{\text{sim}}}\}$ and $\{\lambda^Q, \lambda^K, \lambda^V \in \mathbb{R}^{H_{\text{sim}}}\}$, TINT self-attention with $H_{\text{sim}}$ attention heads and a function $f_{\text{attn}} : \mathbb{R}^{T_{\text{sim}}} \to \mathbb{R}^{T_{\text{sim}}}$ takes a sequence $\{\widehat{e}_t \in \mathbb{R}^{D_{\text{sim}}}\}_{t \le T_{\text{sim}}}$ as input and outputs

$\{\widetilde{\boldsymbol{e}}_t \in \mathbb{R}^{D_{\text{sim}}}\}_{t\le T_{\text{sim}}}$, with

$$\widetilde{\boldsymbol{e}}_t = \textsc{Vectorize}(\{\sum_{j\le T_{\text{sim}}} a_{t,j}^h \widetilde{\boldsymbol{v}}_j^h)_h\}_{h\le H_{\text{sim}}}), \text{ with } a_{t,j}^h = f_{\text{attn}}(\widetilde{\boldsymbol{K}}^h \widetilde{\boldsymbol{q}}_t^h)_j$$
$$\widetilde{\boldsymbol{q}}_t^h = \textsc{Split}_H(\boldsymbol{q}_t)_h + \lambda_h^Q \boldsymbol{W}_Q^p \boldsymbol{p}_t^{\text{TINT}}; \quad \widetilde{\boldsymbol{k}}_t^h = \textsc{Split}_H(\boldsymbol{k}_t)_h + \lambda_h^K \boldsymbol{W}_K^p \boldsymbol{p}_t^{\text{TINT}};$$
$$\widetilde{\boldsymbol{v}}_t^h = \textsc{Split}_H(\boldsymbol{v}_t)_h + \lambda_h^V \boldsymbol{W}_v^p \boldsymbol{p}_t^{\text{TINT}}.$$

Here, $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ denote the query, key, and value vectors at each position $t$, computed as $\boldsymbol{W}_Q^{\text{TINT}}\widehat{\boldsymbol{e}}_t + \boldsymbol{b}_Q^{\text{TINT}}$, $\boldsymbol{W}_K^{\text{TINT}}\widehat{\boldsymbol{e}}_t + \boldsymbol{b}_K^{\text{TINT}}$, and $\boldsymbol{W}_V^{\text{TINT}}\widehat{\boldsymbol{e}}_t + \boldsymbol{b}_V^{\text{TINT}}$ respectively. $\widetilde{\boldsymbol{K}}^h \in \mathbb{R}^{T_{\text{sim}}\times D_{\text{sim}}/H_{\text{sim}}}$ is defined with its rows as $\{\widetilde{\boldsymbol{k}}_t^h\}_{t\le T_{\text{sim}}}$ for all $h \le H_{\text{sim}}$.

$f_{\text{attn}}$ can be either linear or softmax function.

**Bounded parameters and input sequence:** We define a linear self-attention layer to be $B_w$-bounded, if the $\ell_2$ norms of all the parameters are bounded by $B_w$. Going by Definition B.2, this implies

$$\max\{\left\|\boldsymbol{W}_Q^{\text{TINT}}\right\|_2, \left\|\boldsymbol{W}_K^{\text{TINT}}\right\|_2, \left\|\boldsymbol{W}_V^{\text{TINT}}\right\|_2\} \le B_w, \quad \max\{\left\|\boldsymbol{b}_Q^{\text{TINT}}\right\|_2, \left\|\boldsymbol{b}_K^{\text{TINT}}\right\|_2, \left\|\boldsymbol{b}_V^{\text{TINT}}\right\|_2\} \le B_w$$
$$\max\{\left\|\boldsymbol{W}_Q^p\right\|_2, \|\boldsymbol{W}_K^p\|_2, \|\boldsymbol{W}_V^p\|_2\} \le B_w, \quad \max\{\left\|\lambda^Q\right\|_2, \left\|\lambda^K\right\|_2, \left\|\lambda^V\right\|_2\} \le B_w.$$

Furthermore, we define an input sequence $\{\widehat{\boldsymbol{e}}_t\}_{t\le T_{\text{sim}}}$ to $B_x$-bounded, if $\|\widehat{\boldsymbol{e}}_t\|_2 \le B_x$ for all $t$.

Recall from the main paper (Section 3), we used Linear TINT Self-Attention layer to represent the linear operations of the auxiliary model. In the following theorem, we show that a linear attention layer can be represented as a softmax attention layer that uses an additional attention head and an extra token $\boldsymbol{u}$, followed by a linear layer. Therefore, replacing softmax attention with linear attention does not deviate too far from the canonical transformer. We use the Linear TINT Self-Attention layers in several places throughout the model.

**Theorem B.3.** *For any $B_w > 0$, consider a $B_w$-bounded linear self-attention layer that returns $\{\widetilde{\boldsymbol{e}}_t^{linear} \in \mathbb{R}^D_{sim}\}_{t\le T_{sim}}$ on any input $\{\widehat{\boldsymbol{e}}_t \in \mathbb{R}^D_{sim}\}_{t\le T_{sim}}$. Consider a softmax self-attention layer with $2H_{sim}$ attention heads and an additional token $\boldsymbol{u} \in \mathbb{R}^{2D_{sim}}$ such that for any $B_x$-bounded input $\{\widehat{\boldsymbol{e}}_t\}_{t\le T_{sim}}$, it takes a modified input sequence $\{\bar{\boldsymbol{e}}_1, \cdots, \bar{\boldsymbol{e}}_{T_{sim}}, \boldsymbol{u}\}$, and returns $\{\widetilde{\boldsymbol{e}}_t^{softmax} \in \mathbb{R}^{2D_{sim}}\}_{t\le T_{sim}}$. Each modified input token $\bar{\boldsymbol{e}}_t \in \mathbb{R}^{2D_{sim}}$ is obtained by concatenating additional 0s to $\widehat{\boldsymbol{e}}_t$. Then, for any $B_x > 0$, and $\epsilon \le \mathcal{O}(T_{sim}^{-2} B_w^{-5} B_x^{-5})$, there exists $\boldsymbol{W}_O \in \mathbb{R}^{D_{sim}\times 2D_{sim}}$ and such a softmax self-attention layer such that*

$$\left\|\boldsymbol{W}_O \widetilde{\boldsymbol{e}}_t^{softmax} - \widetilde{\boldsymbol{e}}_t^{linear}\right\|_2 \le \mathcal{O}(\sqrt{\epsilon}),$$

*for all $t \le T_{sim}$.*

*Proof.* Consider an input sequence $\{\boldsymbol{x}_t\}_{t\le T_{\text{sim}}}$. Let the attention scores of any linear head $h \le H_{\text{sim}}$ in the linear attention layer be given by $\{a_{t,j}^h\}_{j\le T_{\text{sim}}}$, at any given position $t$. Additionally, let the value vectors for the linear attention be given by $\boldsymbol{v}_t$. To repeat our self-attention definition, the output of the attention layer at any position $t$ is given by $\textsc{Vectorize}(\{\widetilde{\boldsymbol{e}}_t^{linear,h}\}_{h\le H_{\text{sim}}})$, where

$$\widetilde{\boldsymbol{e}}_t^{linear,h} = \sum_{j\le T_{\text{sim}}} a_{t,j}^h \boldsymbol{v}_j^h.$$

Under our assumption, $B_w$ denotes the maximum $\ell_2$ norm of all the parameters in the linear self-attention layer and $B_x$ the maximum $\ell_2$ norm in the input sequence, i.e. $\max_{t\le T_{\text{sim}}} \|\boldsymbol{x}_t\|_2 \le B_x$. With a simple application of Cauchy-Schwartz inequality, we can show that $\max_{j\le T_{\text{sim}}} |a_{t,j}^h| \le \mathcal{O}(B_w^2 B_x^2)$, and $\max_{t\le T_{\text{sim}}} \left\|\boldsymbol{v}_t^h\right\|_2 \le \mathcal{O}(B_w B_x)$.

For $\epsilon \le \mathcal{O}(T_{\text{sim}}^{-10/9} B_w^{-40/9} B_x^{-40/9})$, we can then use Lemma B.4 to represent for each $t, j \le T_{\text{sim}}$,

$$a_{t,j}^h = \frac{\epsilon^{-3} e^{\epsilon a_{t,j}}}{\sum_{t'\le T_{\text{sim}}} e^{\epsilon a_{t,t'}^h} + e^{-2\log\epsilon}} - \epsilon^{-1} + \mathcal{O}\left(\epsilon(T_{\text{sim}} + a_{t,j}^h)\right)$$
$$:= \epsilon^{-3}\text{softmax}\left(\{\epsilon a_{t,1}^h, \epsilon a_{t,2}^h, \cdots, \epsilon a_{t,T_{\text{sim}}}^h, -2\log\epsilon\}\right)_j - \epsilon^{-1} + \mathcal{O}\left(\epsilon^{0.9}\right).$$

**Softmax attention construction:** We define $\boldsymbol{u}$, and the query and key parameters of the softmax attention layer such that for the first $H_{\text{sim}}$ attention heads, the query-key dot products for all the attention heads between any pairs $\{(\bar{\boldsymbol{e}}_t, \bar{\boldsymbol{e}}_j)\}_{t,j\leq T_{\text{sim}}}$ is given by $\{\epsilon a^h_{t,j}\}_{h\leq H_{\text{sim}}}$, while being $-2\log\epsilon$ between $\boldsymbol{u}$ and any token $\bar{\boldsymbol{e}}_t$, with $t \leq T_{\text{sim}}$. For the rest of $H_{\text{sim}}$ attention heads, the attention scores are uniformly distributed across all pairs of tokens (attention score between any pair of tokens is given by $\frac{1}{T_{\text{sim}}+1}$).

We set the value parameters of the softmax attention layer such that at any position $t \leq T_{\text{sim}}$, the value vector is given by $\text{VECTORIZE}(\{\epsilon^{-3}\boldsymbol{v}_t, \boldsymbol{v}_t\})$. The value vector returned for $\boldsymbol{u}$ contains all 0s.

**Softmax attention computation:** Consider an attention head $h \leq H_{\text{sim}}$ in the softmax attention layer now. The output of the attention head at any position $t \leq T_{\text{sim}}$ is given by

$$\begin{aligned}\widetilde{\boldsymbol{e}}^{softmax,h}_t &= \sum_{j\leq T_{\text{sim}}} \text{softmax}\left(\{\epsilon a^h_{t,1}, \epsilon a^h_{t,2}, \cdots, \epsilon a^h_{t,T_{\text{sim}}}, -2\log\epsilon\}\right)_j \epsilon^{-3}\boldsymbol{v}^h_j \\ &= \sum_{j\leq T_{\text{sim}}} \left(a^h_{t,j} + \epsilon^{-1} + \mathcal{O}(\epsilon^{0.9})\right)\boldsymbol{v}^h_j.\end{aligned}$$

This has an additional $\sum_{j\leq T_{\text{sim}}}\left(\epsilon^{-1} + \mathcal{O}(\epsilon^{0.9})\right)\boldsymbol{v}^h_j$, compared to $\widetilde{\boldsymbol{e}}^{linear,h}_t$. However, consider the output of the attention head $H_{\text{sim}} + h$ at the same position:

$$\widetilde{\boldsymbol{e}}^{softmax,H_{\text{sim}}+h}_t = \frac{1}{T_{\text{sim}}+1}\sum_{j\leq T_{\text{sim}}}\boldsymbol{v}^h_j.$$

Hence, we can use the output matrix $\boldsymbol{W}_O$ to get $\widetilde{\boldsymbol{e}}^{softmax,h}_t - \frac{T_{\text{sim}}+1}{\epsilon}\widetilde{\boldsymbol{e}}^{softmax,H_{\text{sim}}+h}_t = \sum_{j\leq T_{\text{sim}}}\left(a^h_{t,j} + \mathcal{O}(\epsilon^{0.9})\right)\boldsymbol{v}^h_j$. The additional term $\mathcal{O}(\epsilon^{0.9})\sum_{j\leq T_{\text{sim}}}\boldsymbol{v}^h_j$ can be further shown to be $\mathcal{O}(\epsilon^{0.5})$ small with the assumed bound of $\epsilon$, since each $\boldsymbol{v}^h_j$ is atmost $\mathcal{O}(B_w B_x)$ in $\ell_2$ norm with a Cauchy Schwartz inequality. □

**Lemma B.4.** *For $\epsilon > 0$, $B > 0$, and a sequence $\{a_1, a_2, \cdots, a_T\}$ with each $a_i \in \mathbb{R}$ and $|a_i| \leq B$, the following holds true for all $i \leq T$,*

$$\frac{\epsilon^{-3}e^{\epsilon a_i}}{\sum_{t'\leq T} e^{\epsilon a_{t'}} + e^{-2\log\epsilon}} = a_i + \frac{1}{\epsilon} + \mathcal{O}\left(\epsilon^{0.9}\right),$$

*provided $\epsilon \leq \mathcal{O}(T^{-10/9}B^{-20/9})$.*

*Proof.* We will use the following first-order Taylor expansions:

$$e^x = 1 + x + \mathcal{O}(x^2). \tag{4}$$

$$\frac{1}{1+x} = 1 - \mathcal{O}(x). \tag{5}$$

Hence, for any $x \ll 1$, $x \approx e^x - 1$.

Simplifying the L.H.S. of the desired bound, we have

$$\begin{aligned}\frac{\epsilon^{-3}e^{\epsilon a_i}}{\sum_{t'\leq T} e^{\epsilon a_{t'}} + e^{-2\log\epsilon}} &= \frac{\epsilon^{-3}(1 + \epsilon a_i + \mathcal{O}(\epsilon^2 a_i^2))}{\sum_{t'\leq T}(1 + \epsilon a_{t'} + \mathcal{O}(\epsilon^2 a_{t'}^2)) + e^{-2\log\epsilon}} && (6)\\ &= \frac{\epsilon^{-1} + a_i + \mathcal{O}(\epsilon a_i^2)}{\sum_{t'\leq T}(\epsilon^2 + \epsilon^3 a_{t'} + \mathcal{O}(\epsilon^4 a_{t'}^2)) + 1} && (7)\\ &= \left(\epsilon^{-1} + a_i + \mathcal{O}(\epsilon a_i^2)\right)\left(1 + \mathcal{O}(\epsilon^2 T)\right) && (8)\\ &= \epsilon^{-1} + a_i + \mathcal{O}(\epsilon T + a_i^2 T\epsilon^2 + a_i^2 T\epsilon^3 + \epsilon a_i^2) = \epsilon^{-1} + a_i + \mathcal{O}(\epsilon^{0.9}).\end{aligned}$$

We used taylor expansion of exponential function( Equation (4) ) in Equation (6) to get Equation (7), and taylor expansion of inverse function(Equation (5)) to get Equation (8) from Equation (7). Furthermore, with the lower bound assumption on $\epsilon$, $\sum_{t'\leq T}(\epsilon^2 + \epsilon^3 a_{t'} + \mathcal{O}(\epsilon^4 a_{t'}^2))$ can be shown to be atmost $3\epsilon^2 T$, which amounts to $\mathcal{O}(\epsilon^2 T)$ error in Equation (8). The final error bound has again been simplified using the lower bound assumption on $\epsilon$. □

### B.1. Simulating Multiplication from (Akyurek et al., 2022)

We refer to the multiplication strategy of (Akyurek et al., 2022) at various places.

**Lemma B.5.** *[Lemma 4 in (Akyurek et al., 2022)] The $GeLU$ (Hendrycks and Gimpel, 2016) nonlinearity can be used to perform multiplication: specifically,*

$$\sqrt{\pi/2}(GeLU(x+y) - GeLU(x) - GeLU(y)) = xy + \mathcal{O}(x^3 + y^3).$$

Thus, to represent an element-wise product or a dot product between two sub-vectors in a token embedding, we can use a MLP with a $GeLU$ activation.

## C. Linear layer

In the main paper, we defined the linear layer without the bias term for simplicity (Definition 3.1). In this section, we will redefine the linear layer with the bias term and present a comprehensive construction of the Linear Forward module.

**Definition C.1** (Linear layer)**.** For a weight $\boldsymbol{W} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$, a linear layer takes $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.

In the discussions below, we consider a linear layer in the auxiliary model with parameters $\{\boldsymbol{W}, \boldsymbol{b}\}$ that takes in input sequence $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T_{\text{aux}}}$ and outputs $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{T_{\text{aux}}}$, with $\boldsymbol{y}_t = \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}$ for each $t \le T_{\text{aux}}$. Since this involves a token-wise operation, we will present our constructed modules with a general token position $t$ and the prefix tokens $\{\boldsymbol{v}_j\}$.

**TINT Linear Forward module** Continuing our discussion from Section 3, we represent $S$ stacked rows of $\boldsymbol{W}$ as a prefix embedding. In addition, we store the bias $\boldsymbol{b}$ in the first prefix embedding ($\boldsymbol{v}_1$).

Using a set of $S'$ unique attention heads in a TINT attention module (Definition B.2), we copy the bias $\boldsymbol{b}$ to respective token embeddings and use a TINT linear layer to add the biases to the final output.

**Auxiliary's backpropagation through linear layer** For a linear layer as defined in Definition C.1, the linear backpropagation layer takes in the loss gradient w.r.t. output ($\partial_{\boldsymbol{y}}$) and computes the loss gradient w.r.t. input ($\partial_{\boldsymbol{x}}$).

**Definition C.2** (Linear backpropagation )**.** For a weight $\boldsymbol{W} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ , the linear backpropagation layer takes $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\partial_{\boldsymbol{x}} = \boldsymbol{W}^\top \partial_{\boldsymbol{y}}$.

**TINT Linear backpropagation module** This module will aim to simulate the auxiliary's linear backpropagation. The input embedding $\boldsymbol{e}_t$ to this module will contain the gradient of the loss w.r.t. $\boldsymbol{y}_t$, i.e. $\partial_{\boldsymbol{y}_t}$. As given in Definition C.2, this module will output the gradient of the loss w.r.t. $\boldsymbol{x}_t$, given by $\partial_{\boldsymbol{x}_t} = \boldsymbol{W}^\top \partial_{\boldsymbol{y}_t}$.

We first use the residual connection to copy the prefix embeddings $\{\boldsymbol{v}_j\}$ (i.e., the rows of $\boldsymbol{W}$) from the forward propagation module. A straightforward construction would be to use the Linear Forward module but with the columns of $\boldsymbol{W}$ stored in the prefix tokens, thereby simulating multiplication with $\boldsymbol{W}^\top$. However, such a construction requires applying attention to the prefix tokens, which increases the size of the construction substantially.

We instead perform the operation more efficiently by splitting it across attention heads. In particular, once we view the operation as $\partial_{\boldsymbol{x}_t} = \sum_i (\partial_{\boldsymbol{y}_t})_i \, \boldsymbol{w}_i$, we can see that the attention score between the current token and the prefix token containing $\boldsymbol{w}_i$ must be $(\partial_{\boldsymbol{y}_t})_i$. Using value vectors as rows of $\boldsymbol{W}$ returns the desired output. Similar to the Linear Forward module, we shard the weights into $S'$ parts to parallelize across more attention heads. Please see Figure 4.

**Auxiliary's linear descent update** Finally, the linear descent layer updates the weight and the bias parameters using a batch of inputs $\{\boldsymbol{x}_t\}_{t \le T_{\text{aux}}}$ and the loss gradient w.r.t. the corresponding outputs $\{\partial_{\boldsymbol{y}_t}\}_{t \le T_{\text{aux}}}$.

**Definition C.3** (Linear descent)**.** For a weight $\boldsymbol{W} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and a bias $\boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$, the linear descent layer takes in a batch of inputs $\{\boldsymbol{x}_t \in \mathbb{R}^D_{\text{aux}}\}_{t \le T_{\text{aux}}}$ and gradients $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^D_{\text{aux}}\}_{t \le T_{\text{aux}}}$ and updates the parameters as follows:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \sum_{t \le T_{\text{aux}}} \partial_{\boldsymbol{y}_t} \boldsymbol{x}_t^\top; \qquad \boldsymbol{b} \leftarrow \boldsymbol{b} - \eta \sum_{t \le T_{\text{aux}}} \partial_{\boldsymbol{y}_t}.$$

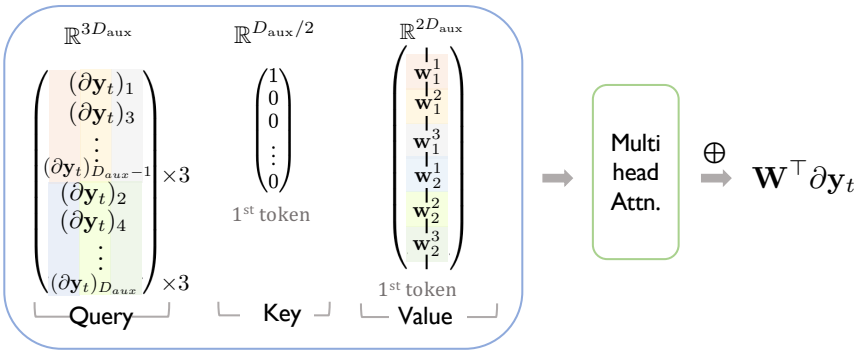


Figure 4: TINT simulates the backward pass of a linear layer as a $H$-head attention layer ($H = 6$ pictured), with the gradient of the loss w.r.t. linear layer output ($\partial_{\boldsymbol{y}_t}$) as the query, the positional one-hot vector of prefix embeddings as the key, and the parameters of the auxiliary model stored in the prefix embeddings as the value. Similar to the Linear Forward module (Figure 2), we distribute the dot product computations across all attention heads by sharding the vectors into $S'$ ($S' = 3$ here) parts. We omitted the identical transformation for query, and value matrices, and permutation-based transformation for key matrix for illustration purposes.

**TINT Linear descent module** The input embedding $\boldsymbol{e}_t$ to this module will contain the gradient of the loss w.r.t. $\boldsymbol{y}_t$, i.e. $\partial_{\boldsymbol{y}_t}$.

As in the Linear backpropagation module, the prefix tokens $\{\boldsymbol{v}_j\}$ will contain the rows of $\boldsymbol{W}$ and $\boldsymbol{b}$, which have been copied from the Linear forward module using residual connections. Since, in addition to the gradients, we also require the input to the linear layer, we will use residual connections to copy the input $\{\boldsymbol{x}_t\}$ to their respective embeddings $\{\boldsymbol{e}_t\}$, from the Linear Forward module. As given in Definition C.3, this module will update $\boldsymbol{W}$ and $\boldsymbol{b}$ using the gradient descent rule.

Focusing on $\boldsymbol{w}_i$, the descent update is given by $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta \sum_t (\partial_{\boldsymbol{y}_t})_i \boldsymbol{x}_t$. For the prefix token $\boldsymbol{v}_j$ that contains $\boldsymbol{w}_i$, the update term $-\eta \sum_t (\partial_{\boldsymbol{y}_t})_i \boldsymbol{x}_t$ can be expressed with an attention head that represents the attention between the prefix token $\boldsymbol{v}_j$ and any token $\boldsymbol{e}_t$ with score $(\partial_{\boldsymbol{y}_t})_i$ and value $-\eta \boldsymbol{x}_t$. The residual connection can then be used to update the weights $\boldsymbol{w}_i$ in $\boldsymbol{v}_j$.

For the bias $\boldsymbol{b}$, the descent update is give by $\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta \sum_t \partial_{\boldsymbol{y}_t}$. With $\boldsymbol{b}$ present in $\boldsymbol{v}_1$, we use one attention head to represent the attention score between prefix token $\boldsymbol{v}_1$ and any token $\boldsymbol{e}_t$ as 1, with the value being $-\eta \partial_{\boldsymbol{y}_t}$. The residual connection can then be used to update the weights $\boldsymbol{b}$ in $\boldsymbol{v}_1$.

The above process can be further parallelized across multiple attention heads, by sharding each weight computation into $S'$ parts. Please see Figure 5.

### C.1. $H_{\text{sim}}$-split operation

We leverage local structure within the linear operations of TINT to make the construction smaller. We build two $H_{\text{sim}}$-split operations to replace all the linear operations. We use $d_{\text{sim}}$ to denote $D_{\text{sim}}/H_{\text{sim}}$ in the following definitions.

**Definition C.4** (Split-wise $H_{\text{sim}}$-split Linear operation)**.** For weight and bias parameters $\boldsymbol{W}^{\text{TINT}} \in \mathbb{R}^{H_{\text{sim}} \times d_{\text{sim}} \times d_{\text{sim}}}$, $\boldsymbol{B}^{\text{TINT}} \in \mathbb{R}^{H_{\text{sim}} \times d_{\text{sim}}}$, this layer takes in input $\boldsymbol{e} \in \mathbb{R}^{D_{\text{sim}}}$ and returns $\widetilde{\boldsymbol{e}} = \text{VECTORIZE}(\widetilde{\boldsymbol{S}} + \boldsymbol{B}^{\text{TINT}})$, with $\widetilde{\boldsymbol{S}} \in \mathbb{R}^{H_{\text{sim}} \times d_{\text{sim}}}$ defined with rows $\{\boldsymbol{W}_h^{\text{TINT}} \text{SPLIT}_{H_{\text{sim}}}(\boldsymbol{e})_h\}_{h \le H_{\text{sim}}}$.

**Definition C.5** (Dimension-wise $H_{\text{sim}}$-split Linear operation)**.** For weight and bias parameters $\boldsymbol{W}^{\text{TINT}} \in \mathbb{R}^{d_{\text{sim}} \times H_{\text{sim}} \times H_{\text{sim}}}$, $\boldsymbol{B}^{\text{TINT}} \in \mathbb{R}^{d_{\text{sim}} \times H_{\text{sim}}}$, this layer takes in input $\boldsymbol{e} \in \mathbb{R}^{D_{\text{sim}}}$, defines $\boldsymbol{S} \in \mathbb{R}^{d_{\text{sim}} \times H_{\text{sim}}}$ with columns $\{\text{SPLIT}_{H_{\text{sim}}}(\boldsymbol{e})_h\}_{h \le H_{\text{sim}}}$, and returns $\widetilde{\boldsymbol{e}} = \text{VECTORIZE}((\widetilde{\boldsymbol{S}} + \boldsymbol{B}^{\text{TINT}})^\top)$, where $\widetilde{\boldsymbol{S}} \in \mathbb{R}^{d_{\text{sim}} \times H_{\text{sim}}}$ is defined with rows $\{\boldsymbol{W}_d^{\text{TINT}} \boldsymbol{s}_d^{\text{TINT}}\}_{d \le d_{\text{sim}}}$.

We find that we can replace all the linear operations with a splitwise $H_{\text{sim}}$-split Linear operation followed by a dimensionwise $H_{\text{sim}}$-split Linear operation, and an additional splitwise $H_{\text{sim}}$-split Linear operation, if necessary. A linear operation on

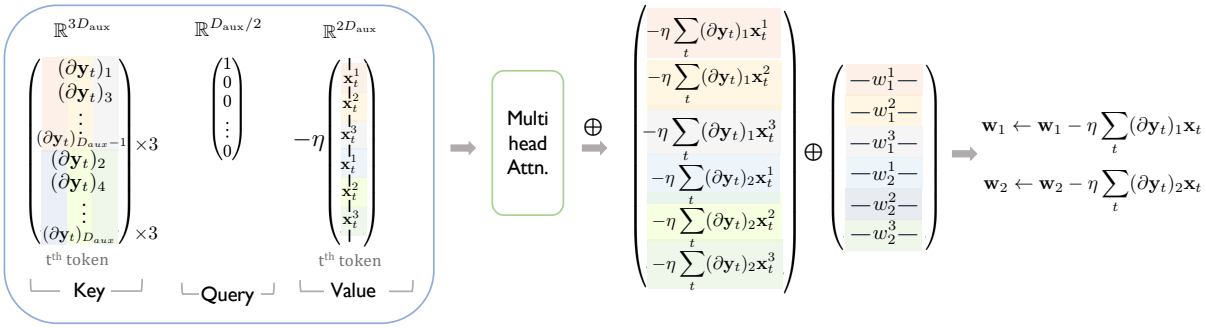


Figure 5: TINT computes the parameter gradients for a linear layer as a $H$-head attention layer ($H = 6$ pictured), with the gradient of the loss w.r.t. linear layer output ($\partial_{\boldsymbol{y}_t}$) as the query, the positional one-hot vector of prefix embeddings as the key, and the input to the linear layer ($\boldsymbol{x}_t$) as the value. The auxiliary model parameters in the prefix embeddings are then updated using a residual connection. Similar to the Linear Forward module (Figure 2), we distribute the dot product computations across all attention heads, by sharding the vectors into $S'$ ($S' = 3$ here) parts. We omitted the identical transformation for query, and value matrices, and permutation-based transformation for key matrix for simplicity.

$D_{\text{sim}}$-dimensional space involves $D_{\text{sim}}^2$ parameters, while its replacement requires $D_{\text{sim}}^2/H_{\text{sim}} + 2D_{\text{sim}}H_{\text{sim}}$ parameters, effectively reducing the total number of necessary parameters by $H_{\text{sim}}$.

We motivate the $H_{\text{sim}}$-split linear operations with an example. We consider the Linear Forward module in Figure 2 for simulating a linear operation with parameters $\boldsymbol{W} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and no biases. For simplicity of presentation, we assume $D_{\text{aux}}$ is divisible by 4. We stack 2 rows of weights per prefix embedding. We distribute the dot-product computation across the $H_{\text{sim}} = 6$ attention heads, by sharding each weight into 3 parts. Since we require to have enough space to store all the sharded computation from the linear attention heads, we require $D_{\text{sim}} = 3D_{\text{aux}}$ (we get 3 values for each of the $D_{\text{aux}}$ weights in $\boldsymbol{W}$). For presentation, for a given vector $\boldsymbol{v} \in \mathbb{R}^{D_{\text{aux}}}$, we represent $\text{SPLIT}_3(\boldsymbol{v})_i$ by $\boldsymbol{v}^i$ for all $1 \le i \le 3$.

Now, consider the final linear operation responsible for combining the output of the attention heads. The output, after the linear operation, should contain $\boldsymbol{W}\boldsymbol{x}_t$ in the first $D_{\text{aux}}$ coordinates. At any position $t$, if we stack the output of the linear attention heads as rows of a matrix $\boldsymbol{S}_t \in \mathbb{R}^{H_{\text{sim}} \times D_{\text{sim}}/H_{\text{sim}}}$ we get

$$\boldsymbol{S}_t = \begin{bmatrix} \langle \boldsymbol{w}_1^1, \boldsymbol{x}_t^1 \rangle & \langle \boldsymbol{w}_3^1, \boldsymbol{x}_t^1 \rangle & \langle \boldsymbol{w}_5^1, \boldsymbol{x}_t^1 \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}-1}^1, \boldsymbol{x}_t^1 \rangle \\ \langle \boldsymbol{w}_1^2, \boldsymbol{x}_t^2 \rangle & \langle \boldsymbol{w}_3^2, \boldsymbol{x}_t^2 \rangle & \langle \boldsymbol{w}_5^2, \boldsymbol{x}_t^2 \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}-1}^2, \boldsymbol{x}_t^2 \rangle \\ \langle \boldsymbol{w}_1^3, \boldsymbol{x}_t^3 \rangle & \langle \boldsymbol{w}_3^3, \boldsymbol{x}_t^3 \rangle & \langle \boldsymbol{w}_5^3, \boldsymbol{x}_t^3 \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}-1}^3, \boldsymbol{x}_t^3 \rangle \\ \langle \boldsymbol{w}_2^1, \boldsymbol{x}_t^1 \rangle & \langle \boldsymbol{w}_4^1, \boldsymbol{x}_t^1 \rangle & \langle \boldsymbol{w}_6^1, \boldsymbol{x}_t^1 \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}}^1, \boldsymbol{x}_t^1 \rangle \\ \langle \boldsymbol{w}_2^2, \boldsymbol{x}_t^2 \rangle & \langle \boldsymbol{w}_4^2, \boldsymbol{x}_t^2 \rangle & \langle \boldsymbol{w}_6^2, \boldsymbol{x}_t^2 \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}}^2, \boldsymbol{x}_t^2 \rangle \\ \langle \boldsymbol{w}_2^3, \boldsymbol{x}_t^3 \rangle & \langle \boldsymbol{w}_4^3, \boldsymbol{x}_t^3 \rangle & \langle \boldsymbol{w}_6^3, \boldsymbol{x}_t^3 \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}}^3, \boldsymbol{x}_t^3 \rangle \end{bmatrix}$$

Note that for each $j \le D_{\text{aux}}$, we have $\langle \boldsymbol{w}_j, \boldsymbol{x}_t \rangle = \sum_{i=1}^{3} \langle \boldsymbol{w}_j^i, \boldsymbol{x}_t^i \rangle$. Thus, with a column-wise linear operation on $\boldsymbol{S}_t$, we can sum the relevant elements in each column to get

$$\boldsymbol{S}_t^{col} = \begin{bmatrix} \langle \boldsymbol{w}_1, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_3, \boldsymbol{x}_t \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}/2-1}, \boldsymbol{x}_t \rangle & 0 & 0 & \cdots & 0 \\ \langle \boldsymbol{w}_2, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_4, \boldsymbol{x}_t \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}/2}, \boldsymbol{x}_t \rangle & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \langle \boldsymbol{w}_{D_{\text{aux}}/2+1}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}/2+3}, \boldsymbol{x}_t \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}-1}, \boldsymbol{x}_t \rangle \\ 0 & 0 & \cdots & 0 & \langle \boldsymbol{w}_{D_{\text{aux}}/2+2}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}/2+4}, \boldsymbol{x}_t \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}}, \boldsymbol{x}_t \rangle \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

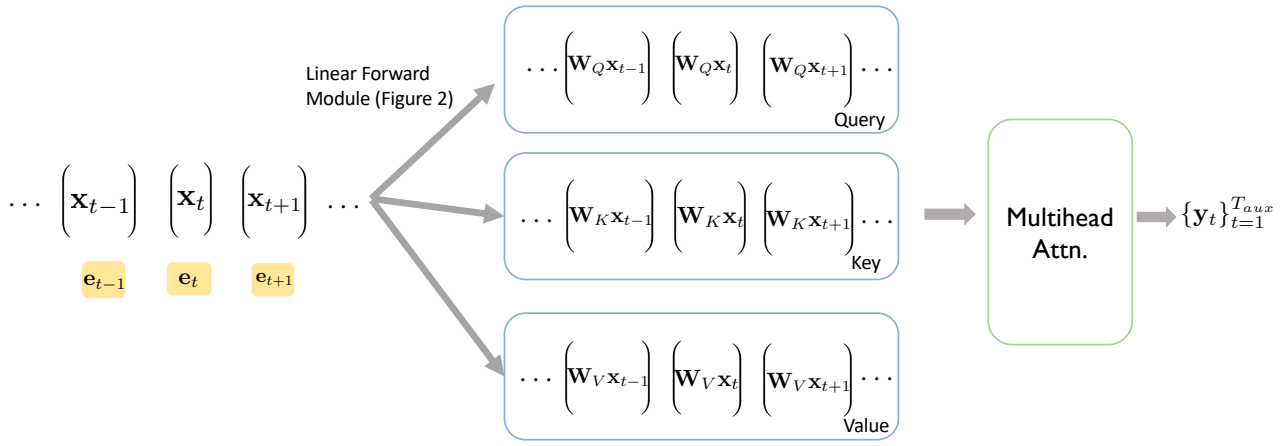


Figure 6: TINT simulates the forward pass of a self-attention layer of the auxiliary model with a Linear Forward module (Figure 2) and a TINT softmax attention layer (Definition B.2). The Linear Forward module computes the query, key, and value vectors using a Linear Forward module on the current embeddings, changing the prefix embeddings to correspond to $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, and $\boldsymbol{W}_K$ respectively.

A row-wise linear operation on $\boldsymbol{S}_t^{col}$ can space out the non-zero elements in the matrix and give us

$$\boldsymbol{S}_t^{row} = \begin{bmatrix} \langle \boldsymbol{w}_1, \boldsymbol{x}_t \rangle & 0 & \langle \boldsymbol{w}_3, \boldsymbol{x}_t \rangle & 0 & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}/2-1}, \boldsymbol{x}_t \rangle & 0 \\ 0 & \langle \boldsymbol{w}_2, \boldsymbol{x}_t \rangle & 0 & \langle \boldsymbol{w}_4, \boldsymbol{x}_t \rangle & \cdots & 0 & \langle \boldsymbol{w}_{D_{\text{aux}}/2}, \boldsymbol{x}_t \rangle \\ \langle \boldsymbol{w}_{D_{\text{aux}}/2+1}, \boldsymbol{x}_t \rangle & 0 & \langle \boldsymbol{w}_{D_{\text{aux}}/2+3}, \boldsymbol{x}_t \rangle & 0 & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}-1}, \boldsymbol{x}_t \rangle & 0 \\ 0 & \langle \boldsymbol{w}_{D_{\text{aux}}/2+2}, \boldsymbol{x}_t \rangle & 0 & \langle \boldsymbol{w}_{D_{\text{aux}}/2+4}, \boldsymbol{x}_t \rangle & \cdots & 0 & \langle \boldsymbol{w}_{D_{\text{aux}}}, \boldsymbol{x}_t \rangle \\ 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \end{bmatrix}$$

Finally, a column-wise linear operation on $\boldsymbol{S}_t^{row}$ helps to get the non-zero elements in the correct order.

$$\bar{\boldsymbol{S}}_t^{col} = \begin{bmatrix} \langle \boldsymbol{w}_1, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_2, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_3, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_4, \boldsymbol{x}_t \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}/2-1}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}/2}, \boldsymbol{x}_t \rangle \\ \langle \boldsymbol{w}_{D_{\text{aux}}/2+1}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}/2+2}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}/2+3}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}/2+4}, \boldsymbol{x}_t \rangle & \cdots & \langle \boldsymbol{w}_{D_{\text{aux}}-1}, \boldsymbol{x}_t \rangle & \langle \boldsymbol{w}_{D_{\text{aux}}}, \boldsymbol{x}_t \rangle \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

The desired output is then given by $\text{VECTORIZE}(\{\bar{s}_{t,j}^{col}\}_{j=1}^{D_{\text{aux}}})$, which contains $\boldsymbol{W}\boldsymbol{x}_t$ in the first $D_{\text{aux}}$ coordinates. The operations that convert $\boldsymbol{S}_t$ to $\boldsymbol{S}_t^{col}$ and $\boldsymbol{S}_t^{row}$ to $\bar{\boldsymbol{S}}_t^{row}$ represents a split-wise 6-split linear operation, while the operation that converts $\boldsymbol{S}_t^{col}$ to $\boldsymbol{S}_t^{row}$ represents a dimension-wise 6-split linear operation. A naive linear operation on the output of the attention heads would require $D_{\text{sim}}^2$ parameters, while its replacement requires $D_{\text{sim}}^2/6$ parameters to represent a dimension-wise 6-split linear operation, and an additional $12D_{\text{sim}}$ parameters to represent the split-wise 6-split linear operations.

## D. Self-attention layer

We first introduce multi-head attention, generalizing single-head attention (Definition B.1).

**Definition D.1** (Auxiliary self-attention with $H_{\text{aux}}$ heads)**.** For query, key, and value weights $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$, a self-attention layer with $H_{\text{aux}}$ attention heads and a function $f_{\text{attn}} : \mathbb{R}^{T_{\text{aux}}} \to \mathbb{R}^{T_{\text{aux}}}$ takes a

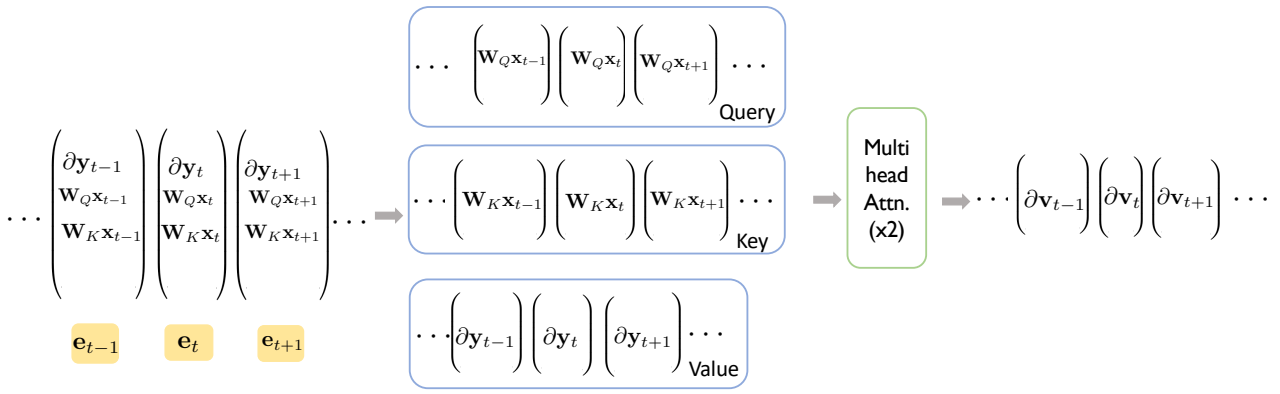


Figure 7: The gradient w.r.t. the value vectors $\{\partial_{\boldsymbol{v}_t}\}$ (Definition D.2) forms the integral component for both TINT self-attention backward and descent update modules. TINT computes $\{\partial_{\boldsymbol{v}_t}\}$ using a softmax attention and a linear attention layer. We first use residual connections to copy the query and key vectors to the current embeddings from the TINT Self-attention Forward module (Figure 6). The softmax attention layer re-computes the attention scores $\{a^h_{t,j}\}$ between all token pairs $\{(t,j)\}$ and stores them in the token embeddings. The linear attention layer uses the one-hot position embeddings of the input tokens as the query to use the transposed attention scores $\{a^h_{j,t}\}$ for all token pairs $\{(t,j)\}$ and use the gradients $\{\partial_{\boldsymbol{y}_t}\}$ as the value vectors to compute $\{\partial_{\boldsymbol{v}_t}\}$.

sequence $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t\le T_{\text{aux}}}$ as input and outputs $\{\boldsymbol{y}_t\}_{t\le T_{\text{aux}}}$, with

$$\boldsymbol{y}_t = \textsc{Vectorize}(\{\sum_{j\le T_{\text{aux}}} a^h_{t,j}\boldsymbol{v}^h_j\}_{h\le H_{\text{aux}}}). \tag{9}$$

$a^h_{t,j}$ is defined as the attention score of head $h$ between tokens at positions $t$ and $j$, and is given by

$$a^h_{t,j} = \text{softmax}(\boldsymbol{K}^h\boldsymbol{q}^h_t)_j. \tag{10}$$

Here, $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ denote the query, key, and value vectors at each position $t$, computed as $\boldsymbol{W}_Q\boldsymbol{x}_t + \boldsymbol{b}_Q$, $\boldsymbol{W}_K\boldsymbol{x}_t + \boldsymbol{b}_K$, and $\boldsymbol{W}_V\boldsymbol{x}_t + \boldsymbol{b}_V$ respectively. In addition, $\boldsymbol{q}^h_t, \boldsymbol{k}^h_t, \boldsymbol{v}^h_t$ denote $\textsc{Split}_{H_{\text{aux}}}(\boldsymbol{q}_t)_h$, $\textsc{Split}_{H_{\text{aux}}}(\boldsymbol{k}_t)_h$, and $\textsc{Split}_{H_{\text{aux}}}(\boldsymbol{v}_t)_h$ respectively for all $t \le T_{\text{aux}}$, and $h \le H_{\text{aux}}$. $\boldsymbol{K}^h \in \mathbb{R}^{T_{\text{aux}}\times D_{\text{aux}}}$ is defined with its rows as $\{\boldsymbol{k}^h_t\}_{t\le T_{\text{aux}}}$ for all $h \le H_{\text{aux}}$.

In the discussions below, we consider a self-attention layer in the auxiliary model with parameters $\{\boldsymbol{W}_Q, \boldsymbol{b}_Q, \boldsymbol{W}_K, \boldsymbol{b}_K, \boldsymbol{W}_V, \boldsymbol{b}_V\}$ that takes in input sequence $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T_{\text{aux}}}$ and outputs $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{T_{\text{aux}}}$, with $\{\boldsymbol{y}_t\}_{t=1}^{T_{\text{aux}}}$ given by (9). As in the definition, $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ denote the query, key, and value vectors for position $t$. We will use TINT self-attention modules in order to simulate the operations on the auxiliary's self-attention layer. To do so, we will need $H_{\text{sim}} \ge H_{\text{aux}}$ in the corresponding TINT self-attention modules.

**TINT Self-attention forward module** The input embedding to this module $\boldsymbol{e}_t$ at each position $t$ will contain $\boldsymbol{x}_t$ in its first $D_{\text{aux}}$ coordinates. The self-attention module can be divided into four sub-operations: Computation of (a) query vectors $\{\boldsymbol{q}_t\}_{t\le T}$, (b) key vectors $\{\boldsymbol{k}_t\}_{t\le T}$, (c) value vectors $\{\boldsymbol{v}_t\}_{t\le T}$, and (d) $\{\boldsymbol{y}_t\}_{t\le T}$ using (9). Please see Figure 6.

- Sub-operations (a): The computation of query vector $\boldsymbol{q}_t := \boldsymbol{W}_Q\boldsymbol{x}_t + \boldsymbol{b}_Q$ at each position $t$ is a linear operation involving parameters $\boldsymbol{W}_Q, \boldsymbol{b}_Q$. Thus, we can first feed in the stacked rows of $\boldsymbol{W}_Q$ and $\boldsymbol{b}_Q$ onto the prefix embeddings $\{\boldsymbol{v}_j\}$. We use a Linear Forward module (Appendix C) on the current embeddings and the prefix embeddings to get embedding $\boldsymbol{e}^q_t$ at each position $t$ that contains $\boldsymbol{q}_t$ in the first $D_{\text{aux}}$ coordinates.
- Sub-operations (b, c): Similar to (a), we feed in the stacked rows of the necessary parameters onto the prefix embeddings $\{\boldsymbol{v}_j\}$, and call two Linear Forward Modules (Appendix C) independently to get embeddings $\boldsymbol{e}^k_t$, and $\boldsymbol{e}^v_t$ containing $\boldsymbol{k}_t$ and $\boldsymbol{v}_t$ respectively.

  We now combine the embeddings $\boldsymbol{e}^q_t$, $\boldsymbol{e}^k_t$, and $\boldsymbol{e}^v_t$ to get an embedding $\boldsymbol{e}_t$ that contain $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ in the first $3D_{\text{aux}}$ coordinates.

- Sub-operation (d): Finally, we call a TINT self-attention module (Definition B.2) on our current embeddings $\{\boldsymbol{e}_t\}_{t\le T}$ to compute $\{\boldsymbol{y}_t\}_{t\le T}$. The query, key, and value parameters in the self-attention module contain sub-Identity blocks that pick out the relevant information from $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ stored in $\boldsymbol{e}_t$.

*Remark:* Sub-operations (a), (b), and (c) can be represented as a single linear operation with a weight $\boldsymbol{W} \in \mathbb{R}^{3D_{\text{aux}} \times D_{\text{aux}}}$ by concatenating the rows of $\{\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V\}$ and a bias $\boldsymbol{b} \in \mathbb{R}^{3D_{\text{aux}}}$ that concatenates $\{\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V\}$. Thus, they can be simulated with a single Linear Forward Module, with $\boldsymbol{W}, \boldsymbol{b}$ fed into the prefix embeddings. However, we decide to separate them in order to limit the number of prefix embeddings and the embedding size. E.g. for GPT-2, $D_{\text{aux}} = 768$. This demands either a $3\times$ increase in the embedding size in TINT or a $3\times$ increase in the number of prefix embeddings. Hence, in order to minimize the parameter cost, we call Linear Forward Module separately to compute $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ at each position $t$.

**Auxiliary's backpropagation through self-attention** For an auxiliary self-attention layer as defined in Definition D.1, the backpropagation layer takes in the loss gradient w.r.t. output ($\{\partial_{\boldsymbol{y}_t}\}_{t\le T_{\text{aux}}}$) and computes the loss gradient w.r.t. input token ($\{\partial_{\boldsymbol{x}_t}\}_{t\le T_{\text{aux}}}$).

**Definition D.2.** [Auxiliary self-attention backpropagation] For query, key, and value weights $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$, the backpropagation layer corresponding to a self-attention layer with $H_{\text{aux}}$ attention heads takes a sequence $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^{D_{\text{aux}}}\}_{t\le T_{\text{aux}}}$ and $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t\le T_{\text{aux}}}$ as input and outputs $\{\partial_{\boldsymbol{x}_t}\}_{t\le T_{\text{aux}}}$, with

$$
\begin{aligned}
\partial_{\boldsymbol{x}_t} &= \boldsymbol{W}_Q^\top \partial_{\boldsymbol{q}_t} + \boldsymbol{W}_K^\top \partial_{\boldsymbol{k}_t} + \boldsymbol{W}_V^\top \partial_{\boldsymbol{v}_t}, \quad \text{with} \\
\partial_{\boldsymbol{q}_t} &= \text{VECTORIZE}(\{\sum_j a^h_{t,j}((\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h)[\boldsymbol{k}_j^h - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}]\}_{h\le H_{\text{aux}}}); \\
\partial_{\boldsymbol{k}_t} &= \text{VECTORIZE}(\{\sum_j a^h_{j,t} \boldsymbol{q}_j^h[(\partial_{\boldsymbol{y}_j^h})^\top (\boldsymbol{v}_t^h - \sum_{j'} a^h_{j,j'} \boldsymbol{v}^h_{j'})]\}_{h\le H_{\text{aux}}}); \\
\partial_{\boldsymbol{v}_t} &= \text{VECTORIZE}(\{\sum_j a^h_{j,t} \partial_{\boldsymbol{y}_j^h}\}_{h\le H_{\text{aux}}})
\end{aligned}
$$

Here, $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ refer to query, key, and value vectors at each position $t$, with the attention scores $\{a^h_{t,j}\}_{t,j\le T_{\text{aux}}, h\le H_{\text{aux}}}$.

**Complexity of true backpropagation** The much-involved computation in the above operation is due to the computation of $\partial_{\boldsymbol{q}_t}$ and $\partial_{\boldsymbol{k}_t}$ at each position $t$. For the following discussion, we assume that our current embeddings $\boldsymbol{e}_t$ contain $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$, in addition to the gradient $\partial_{\boldsymbol{y}_t}$. The computation of $\partial_{\boldsymbol{q}_t}$ (and similarly $\partial_{\boldsymbol{k}_t}$) at any position $t$ involves the following sequential computations and the necessary TINT modules.

- $\{\{\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\}_{j\le T_{\text{aux}}}\}_{h\le H_{\text{aux}}}$ with a TINT linear self-attention module (Definition B.2), with atleast $H_{\text{aux}}$ attention heads that represent the attention score between $\boldsymbol{e}_t$ and any other token $\boldsymbol{e}_j$, by $\{(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\}_{h\le H_{\text{aux}}}$.
- Attention scores $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$, which requires a TINT softmax self-attention module (Definition B.2), with at least $H_{\text{aux}}$ heads, that uses the already present $\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\}$ in the current embeddings $\boldsymbol{e}_t$ to re-compute the attention scores.
- $\{a^h_{t,j}(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\}_{h\le H_{\text{aux}}}$ for all $j \le T_{\text{aux}}$ by multiplying the attention scores $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ with $\{(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\}_{h\le H_{\text{aux}}}$ using an MLP layer (Lemma B.5). Furthermore, $\{\sum_j a^h_{t,j} \boldsymbol{k}_j^h\}_{h\le H_{\text{aux}}}$ needs to be computed in parallel as well, with additional attention heads.
- $\partial_{\boldsymbol{y}_t}$ with a TINT linear self-attention module (Definition B.2), with atleast $H_{\text{aux}}$ attention heads that represent the attention score between any token $\boldsymbol{e}_j$ and $\boldsymbol{e}_t$ by $\{a^h_{t,j}(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\}_{h\le H_{\text{aux}}}$, with value vectors given by $\{\boldsymbol{k}_j^h - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}\}_{h\le H_{\text{aux}}}$.

The sequential computation requires the simulator to store $\{\{\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\}_{j\le T_{\text{aux}}}\}_{h\le H_{\text{aux}}}$ and $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ in the token embedding $\boldsymbol{e}_t$, which requires an additional $2T_{\text{aux}}H_{\text{aux}}$ embedding dimension size. To avoid the much-involved computation for the true gradient propagation, we instead only use the gradients w.r.t. $\boldsymbol{v}_t$.

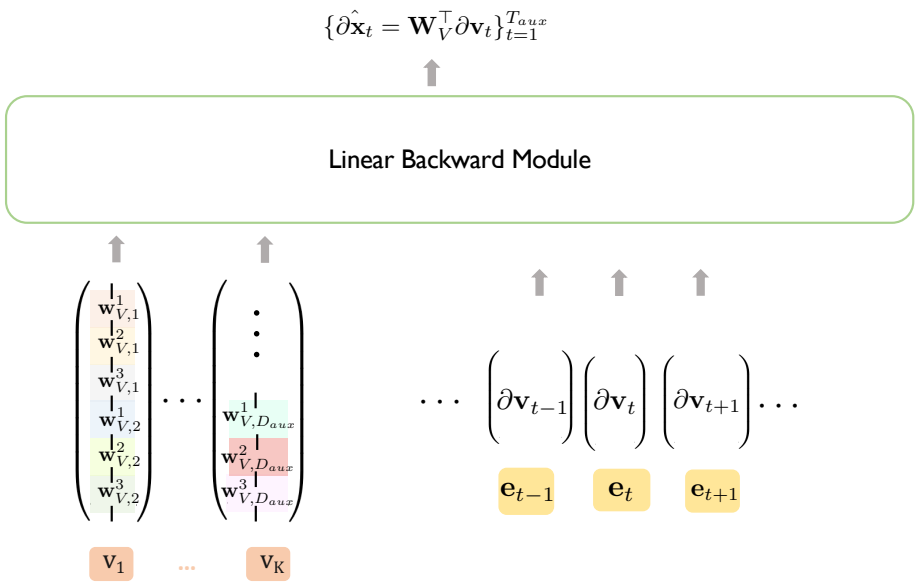


Figure 8: TINT simulates the backward pass of a self-attention layer of the auxiliary model using a Linear Backward module (Figure 4). The input embeddings contain the gradient of the loss w.r.t. the value vectors ($\partial_{\boldsymbol{v}_t}$) computed in Figure 7. The value matrix $\boldsymbol{W}_V$ is encoded in the prefix embeddings. We call the Linear Backward module on this sequence.

**Approximate auxiliary self-attention backpropagation** We formally extend the definition of approximate gradients $\{\partial_{\boldsymbol{x}_t}\}_{t=1}^{T_{\text{aux}}}$ from Definition D.3 to multi-head attention in Definition D.3.

**Definition D.3.** For query, key, and value weights $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$, the approximate backpropagation layer corresponding to a self-attention layer with $H_{\text{aux}}$ attention heads takes a sequence $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ and $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ as input and outputs $\{\partial_{\boldsymbol{x}_t} := \textsc{Vectorize}(\{\partial_{\boldsymbol{x}_t^h}\}_{h \le H_{\text{aux}}})\}_{t \le T_{\text{aux}}}$, with

$$\widehat{\partial_{\boldsymbol{x}_t}} = \boldsymbol{W}_V^\top \partial_{\boldsymbol{v}_t}, \quad \text{where } \partial_{\boldsymbol{v}_t} = \textsc{Vectorize}(\{\sum_j a_{j,t}^h \partial_{\boldsymbol{y}_j^h}\}_{h \le H_{\text{aux}}})$$

Here, $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ refer to query, key, and value vectors at each position $t$, as defined in Definition D.1, with the attention scores $\{a_{t,j}^h\}_{t,j \le T_{\text{aux}}, h \le H_{\text{aux}}}$ defined in Equation (10).

In the upcoming theorem, we formally show that if on a given sequence $\{\boldsymbol{x}_t\}_{t \le T_{\text{aux}}}$, for all token positions all the attention heads in a self-attention layer primarily attend to a single token, then the approximate gradient $\widehat{\partial_{\boldsymbol{x}_t}}$ is close to the true gradient $\partial_{\boldsymbol{x}_t}$ at each position $t$.

**Definition D.4** ($\varepsilon$-hard attention head)**.** For the Self-Attention layer of $H_{\text{aux}}$ heads in Definition D.1, on a given input sequence $\{\boldsymbol{x}_t\}_{t=1}^{T_{\text{aux}}}$, an attention head $h \le H_{\text{aux}}$ is defined to be $\varepsilon$-hard on the input sequence, if for all positions $t \le T_{\text{aux}}$, there exists a position $t_0 \le T_{\text{aux}}$ such that $a_{t,t_0}^h \ge 1 - \varepsilon$.

**Theorem D.5.** *With the notations in Definitions D.1 to D.3, if on a given input sequence $\{\boldsymbol{x}_t\}_{t=1}^{T_{aux}}$, with its query, key, and value vectors $\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\}_{t=1}^{T_{aux}}$, all the $H_{aux}$ attention heads are $\varepsilon$-hard for some $\varepsilon > 0$, then for a given sequence of gradients $\{\partial_{\boldsymbol{y}_t}\}_{t=1}^{T_{aux}}$,*

$$\|\partial_{\boldsymbol{q}_t}\|_2, \|\partial_{\boldsymbol{k}_t}\|_2 \le \mathcal{O}(\varepsilon B_x^2 B_w^2 B_y), \quad \textit{for all } t \le T_{aux},$$

*where* $B_x = \max_{t \le T_{aux}} \|\boldsymbol{x}_t\|_2$, $B_y = \max_{t \le T_{aux}} \|\partial_{\boldsymbol{y}_t}\|_2$, *and* $B_w = \max\{\|\boldsymbol{W}_K\|_2, \|\boldsymbol{W}_Q\|_2, \|\boldsymbol{W}_V\|_2, \|\boldsymbol{b}_V\|_2, \|\boldsymbol{b}_K\|_2, \|\boldsymbol{b}_V\|_2\}$.

*This implies, for each position $t$,* $\left\|\widehat{\partial_{\boldsymbol{x}_t}} - \partial_{\boldsymbol{x}_t}\right\|_2 \le \mathcal{O}(\varepsilon B_x^2 B_w^3 B_y)$.

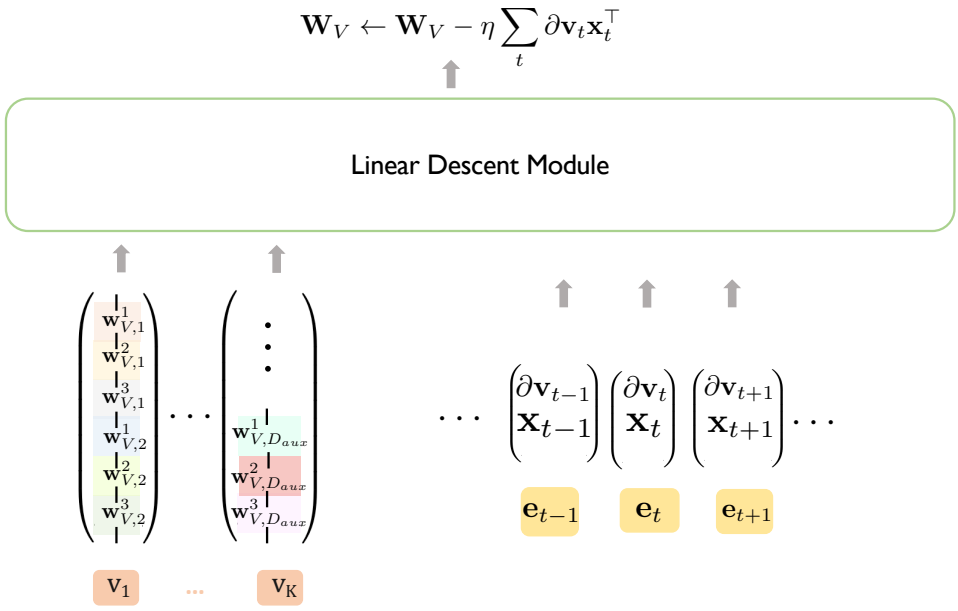


Figure 9: TINT simulates the backward pass of the self-attention layer in the auxiliary model by employing the Linear Descent module (Figure 5). The input embeddings consist of the gradient of the loss with respect to the value vectors ($\partial_{\boldsymbol{v}_t}$) computed in Figure 7. Additionally, we incorporate a residual connection to copy the input from the Self-attention Forward module (Figure 6) into $\boldsymbol{x}_t$. Before invoking the Linear Descent module, we represent the value parameters ($\boldsymbol{W}_V$) into the prefix embeddings. TINT simulates the backward pass of a self-attention layer of the auxiliary model using a Linear Descent module (Figure 5).

**TINT Self-attention backpropagation module** The input embeddings $\boldsymbol{e}_t$ contain $\partial_{\boldsymbol{y}_t}$ in the first $D_{\text{aux}}$ coordinates. Since we require to re-compute the attention scores $\{a^h_{t,j}\}_{j\le T_{\text{aux}}, h\le H_{\text{aux}}}$, we need to copy the query, key, and value vectors $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ from the TINT self-attention Forward module at each position $t$. Furthermore, we use the residual connection to copy the prefix embeddings $\{\boldsymbol{v}_j\}$, which contain the rows of $\boldsymbol{W}_V$, from the TINT self-attention Forward module.

The operation can be divided into three sub-operations: Computing (a) attention scores $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ for all $j \le T_{\text{aux}}$, at each position $t$, (b) $\partial_{\boldsymbol{v}_t}$ from $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ and $\partial_{\boldsymbol{y}_t}$, and (c) $\widehat{\partial_{\boldsymbol{x}_t}}$ from $\partial_{\boldsymbol{v}_t}$.

- Sub-operation (a): Since, the current embeddings $\boldsymbol{e}_t$ contain $\boldsymbol{q}_t, \boldsymbol{k}_t$, we can simply call a self-attention attention module to compute the attention scores $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ for all $j \le T$ and store them in the current embeddings. We further retain $\partial_{\boldsymbol{y}_t}$ and $\boldsymbol{v}_t$ for further operations using residual connections.
- Sub-operation (b): With the current embeddings $\boldsymbol{e}_t$ containing the attention scores $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ for all $j \le T$, and the gradient $\partial_{\boldsymbol{y}_t}$, we can compute $\partial_{\boldsymbol{v}_t}$ using a TINT linear self-attention module with atleast $H_{\text{aux}}$ attention heads, that represent the attention scores between tokens $\boldsymbol{e}_t$ and $\boldsymbol{e}_j$ for any $j$ as $\{a^h_{j,t}\}_{h\le H_{\text{aux}}}$ and use $\text{SPLIT}_{H_{\text{aux}}}(\partial_{\boldsymbol{y}_t})$ as their value vectors.
- Sub-operation (c): And finally, the computation of $\widehat{\partial_{\boldsymbol{x}_t}}$ is identical to the backpropagation through a linear layer, with parameters $\boldsymbol{W}_V$ and $\boldsymbol{b}_V$. Hence, we call a Linear backpropagation module on the current embeddings, that contain $\partial_{\boldsymbol{y}_t}$ and the prefix embeddings that contain $\boldsymbol{W}_V$ and $\boldsymbol{b}_V$.

**Separating sub-operations (a) and (b)** The operation for computing $\partial_{\boldsymbol{v}_t}$ in Definition D.3 looks very similar to the computation of $\boldsymbol{y}_t$ in Equation (9). However, the major difference is that instead of the attention scores being $\{a^h_{t,j}\}_{h\le H_{\text{aux}}}$ between token $t$ and any token $j$, we need the attention scores to be $\{a^h_{j,t}\}_{h\le H_{\text{aux}}}$. Thus, unless our model allows a transpose operation on the attention scores, we need to first store them in our embeddings and then use an additional self-attention module that can pick the right attention scores between tokens using position embeddings. Please see Figure 8.

**Auxiliary's value descent update** Similar to the complexity of true backpropagation, the descent updates for $\boldsymbol{W}_Q, \boldsymbol{b}_Q, \boldsymbol{W}_K, \boldsymbol{b}_K$ are quite expensive to express with the transformer layers. Hence, we focus simply on updating on $\boldsymbol{W}_V, \boldsymbol{b}_V$, while keeping the others fixed.

**Definition D.6** (Auxiliary self-attention value descent)**.** For query, key, and value weights $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$, the value descent layer corresponding to a self-attention layer with $H_{\text{aux}}$ attention heads and any function $f_{\text{attn}} : \mathbb{R}^{T_{\text{aux}}} \to \mathbb{R}^{T_{\text{aux}}}$ takes in a batch of gradients $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ and inputs $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ and updates $\boldsymbol{W}_V, \boldsymbol{b}_V$ as follows:

$$\boldsymbol{W}_V \leftarrow \boldsymbol{W}_V - \eta \sum_{t \le T_{\text{aux}}} \partial_{\boldsymbol{v}_t} \boldsymbol{x}_t^\top, \quad \boldsymbol{b}_V \leftarrow \boldsymbol{b}_V - \eta \sum_{t \le T_{\text{aux}}} \partial_{\boldsymbol{v}_t},$$
$$\text{where } \partial_{\boldsymbol{v}_t} = \textsc{Vectorize}(\{\sum_j a_{j,t}^h \partial_{\boldsymbol{y}_j^h}\}_{h \le H_{\text{aux}}})$$

Here, $\boldsymbol{v}_t$ refers to value vectors at each position $t$, as defined in Definition D.1.

**TINT Self-attention descent module** The input embeddings contain $\partial_{\boldsymbol{v}_t}$ in the first $D_{\text{aux}}$ coordinates, from the TINT self-attention backpropagation module. Furthermore, the prefix embeddings $\{\boldsymbol{v}_j\}$ contain the stacked rows of $\boldsymbol{W}_V$ and $\boldsymbol{b}_V$, continuing from the TINT self-attention backpropagation module.

Since we further need the input $\boldsymbol{x}_t$ to the auxiliary self-attention layer under consideration, we use residual connections to copy $\boldsymbol{x}_t$ from the TINT self-attention Forward module at each position $t$.

The updates of $\boldsymbol{W}_V$ and $\boldsymbol{b}_V$ are equivalent to the parameter update in a linear layer, involving gradients $\{\partial_{\boldsymbol{v}_t}\}$ and input $\{\boldsymbol{x}_t\}$. Thus, we call a Linear descent module on the current embeddings and the prefix embeddings to get the updated value parameters. Please see Figure 9.

### D.1. Proofs of theorems and gradient definitions

We restate the theorems and definitions, before presenting their proofs for easy referencing.

**Definition D.2.** [Auxiliary self-attention backpropagation] For query, key, and value weights $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$, the backpropagation layer corresponding to a self-attention layer with $H_{\text{aux}}$ attention heads takes a sequence $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ and $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ as input and outputs $\{\partial_{\boldsymbol{x}_t}\}_{t \le T_{\text{aux}}}$, with

$$\begin{aligned}
\partial_{\boldsymbol{x}_t} &= \boldsymbol{W}_Q^\top \partial_{\boldsymbol{q}_t} + \boldsymbol{W}_K^\top \partial_{\boldsymbol{k}_t} + \boldsymbol{W}_V^\top \partial_{\boldsymbol{v}_t}, \quad \text{with} \\
\partial_{\boldsymbol{q}_t} &= \textsc{Vectorize}(\{\sum_j a_{t,j}^h ((\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h)[\boldsymbol{k}_j^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h]\}_{h \le H_{\text{aux}}}); \\
\partial_{\boldsymbol{k}_t} &= \textsc{Vectorize}(\{\sum_j a_{j,t}^h \boldsymbol{q}_j^h [(\partial_{\boldsymbol{y}_j^h})^\top (\boldsymbol{v}_t^h - \sum_{j'} a_{j,j'}^h \boldsymbol{v}_{j'}^h)]\}_{h \le H_{\text{aux}}}); \\
\partial_{\boldsymbol{v}_t} &= \textsc{Vectorize}(\{\sum_j a_{j,t}^h \partial_{\boldsymbol{y}_j^h}\}_{h \le H_{\text{aux}}})
\end{aligned}$$

Here, $\boldsymbol{q}_t$, $\boldsymbol{k}_t$, and $\boldsymbol{v}_t$ refer to query, key, and value vectors at each position $t$, with the attention scores $\{a_{t,j}^h\}_{t,j \le T_{\text{aux}}, h \le H_{\text{aux}}}$.

*Derivation of gradient in Definition D.2.* Recalling the definition of $\boldsymbol{y}_t$ from Definition D.1,

$$\boldsymbol{y}_t = \textsc{Vectorize}(\{\sum_{j \le T_{\text{aux}}} a_{t,j}^h \boldsymbol{v}_j^h\}_{h \le H_{\text{aux}}}); \quad a_{t,j}^h = \text{softmax}(\boldsymbol{K}^h \boldsymbol{q}_t^h)_j,$$
$$\boldsymbol{q}_t = \boldsymbol{W}_Q \boldsymbol{x}_t + \boldsymbol{b}_Q \quad \boldsymbol{k}_t = \boldsymbol{W}_K \boldsymbol{x}_t + \boldsymbol{b}_K, \quad \boldsymbol{v}_t = \boldsymbol{W}_V \boldsymbol{x}_t + \boldsymbol{b}_V.$$

$\boldsymbol{q}_t^h, \boldsymbol{k}_t^h, \boldsymbol{v}_t^h$ denote $\textsc{Split}_{H_{\text{aux}}}(\boldsymbol{q}_t)_h$, $\textsc{Split}_{H_{\text{aux}}}(\boldsymbol{k}_t)_h$, and $\textsc{Split}_{H_{\text{aux}}}(\boldsymbol{v}_t)_h$ respectively for all $t \le T_{\text{aux}}$, and $h \le H_{\text{aux}}$. $\boldsymbol{K}^h \in \mathbb{R}^{T_{\text{aux}} \times D_{\text{aux}}}$ is defined with its rows as $\{\boldsymbol{k}_t^h\}_{t \le T_{\text{aux}}}$ for all $h \le H_{\text{aux}}$.

We explain the proof for an arbitrary token position $t$. With the application of the chain rule, we have

$$\begin{aligned}
\partial_{\boldsymbol{x}_t} &= (\frac{\partial \boldsymbol{q}_t}{\partial \boldsymbol{x}_t})^\top \partial_{\boldsymbol{q}_t} + (\frac{\partial \boldsymbol{k}_t}{\partial \boldsymbol{x}_t})^\top \partial_{\boldsymbol{k}_t} + (\frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{x}_t})^\top \partial_{\boldsymbol{v}_t} \\
&= \boldsymbol{W}_Q^\top \partial_{\boldsymbol{q}_t} + \boldsymbol{W}_K^\top \partial_{\boldsymbol{k}_t} + \boldsymbol{W}_V^\top \partial_{\boldsymbol{v}_t},
\end{aligned}$$

where the second step follows from the definitions of $\boldsymbol{q}_t, \boldsymbol{k}_t$, and $\boldsymbol{v}_t$ respectively.

**Computation of $\partial_{\boldsymbol{q}_t}$:** With the SPLIT operation of $\boldsymbol{q}_t$ across $H_{\text{aux}}$ heads for the computation of $\boldsymbol{y}_t$, the computation of the backpropagated gradient $\partial_{\boldsymbol{q}_t}$ itself needs to be split across $H_{\text{aux}}$ heads. Furthermore, query vector $\boldsymbol{q}_t$ only affects $\boldsymbol{y}_t$, implying $\frac{\partial \boldsymbol{y}_{t'}}{\partial \boldsymbol{q}_t} = 0$ for any $t' \neq t$. Thus, we have for any head $h \leq H_{\text{aux}}$, if $\boldsymbol{y}_t^h$ represents the output of attention head $h$, given by $\sum_{j \leq T_{\text{aux}}} a_{t,j}^h \boldsymbol{v}_j^h$,

$$
\begin{aligned}
\partial_{\boldsymbol{q}_t^h} &= (\frac{\partial \boldsymbol{y}_t^h}{\partial \boldsymbol{q}_t^h})^\top \partial_{\boldsymbol{y}_t^h} \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_j^h, \partial_{\boldsymbol{y}_t^h} \rangle \frac{\partial a_{t,j}^h}{\partial \boldsymbol{q}_t^h} \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_j^h, \partial_{\boldsymbol{y}_t^h} \rangle \frac{\partial}{\partial \boldsymbol{q}_t^h} \left( \frac{e^{\langle \boldsymbol{k}_j^h, \boldsymbol{q}_t^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_t^h \rangle}} \right) && (11) \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_j^h, \partial_{\boldsymbol{y}_t^h} \rangle \left[ \frac{1}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_t^h \rangle}} \frac{\partial e^{\langle \boldsymbol{k}_j^h, \boldsymbol{q}_t^h \rangle}}{\partial \boldsymbol{q}_t^h} - \left( \frac{e^{\langle \boldsymbol{k}_j^h, \boldsymbol{q}_t^h \rangle}}{(\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_t^h \rangle})^2} \right) \sum_{j' \leq T_{\text{aux}}} \frac{\partial e^{\langle \boldsymbol{k}_{j'}^h, \boldsymbol{q}_t^h \rangle}}{\partial \boldsymbol{q}_t^h} \right] && (12) \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_j^h, \partial_{\boldsymbol{y}_t^h} \rangle \left[ \left( \frac{e^{\langle \boldsymbol{k}_j^h, \boldsymbol{q}_t^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_t^h \rangle}} \right) \boldsymbol{k}_j^h - \left( \frac{e^{\langle \boldsymbol{k}_j^h, \boldsymbol{q}_t^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_t^h \rangle}} \right) \sum_{j' \leq T_{\text{aux}}} \left( \frac{e^{\langle \boldsymbol{k}_{j'}^h, \boldsymbol{q}_t^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_t^h \rangle}} \right) \boldsymbol{k}_{j'}^h \right] && (13) \\
&= \sum_{j \leq T_{\text{aux}}} a_{t,j}^h \langle \boldsymbol{v}_j^h, \partial_{\boldsymbol{y}_t^h} \rangle \left( \boldsymbol{k}_j^h - \sum_{j' \leq T_{\text{aux}}} a_{t,j'}^h \boldsymbol{k}_{j'}^h \right).
\end{aligned}
$$

In Equation (11), we have expanded the definition of $\mathrm{softmax}$ in $a_{t,j}^h := \mathrm{softmax}(\boldsymbol{K}^h \boldsymbol{q}_t^h)_j$ in order to better motivate the derivative of $a_{t,j}^h$ w.r.t. $\boldsymbol{q}_t^h$. Finally, $\partial_{\boldsymbol{q}_t}$ is given by $\text{VECTORIZE}(\{\partial_{\boldsymbol{q}_t^h}\}_{h \leq H_{\text{aux}}})$.

**Computation of $\partial_{\boldsymbol{k}_t}$:** Continuing as the computation of $\partial_{\boldsymbol{q}_t}$, we split the computation of $\partial_{\boldsymbol{k}_t}$ across the $H_{\text{aux}}$ attention heads. However, unlike $\boldsymbol{q}_t$, $\boldsymbol{k}_t$ affects $\boldsymbol{y}_j$ for all $j \leq T_{\text{aux}}$. For any head $h \leq H_{\text{aux}}$, we follow the chain-rule step by step to get

$$
\begin{aligned}
\partial_{\boldsymbol{k}_t^h} &= \sum_{j \leq T_{\text{aux}}} (\frac{\partial \boldsymbol{y}_j^h}{\partial \boldsymbol{k}_t^h})^\top \partial_{\boldsymbol{y}_j^h} = \sum_{j \leq T_{\text{aux}}} \left( \frac{\partial \sum_{j' \leq T_{\text{aux}}} a_{j,j'} \boldsymbol{v}_{j'}^h}{\partial \boldsymbol{k}_t^h} \right)^\top \partial_{\boldsymbol{y}_j^h} \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_t^h, \partial_{\boldsymbol{y}_j^h} \rangle \frac{\partial a_{j,t}^h}{\partial \boldsymbol{k}_t^h} + \sum_{j \leq T_{\text{aux}}} \sum_{j' \leq T_{\text{aux}}; j' \neq t} \langle \boldsymbol{v}_{j'}^h, \partial_{\boldsymbol{y}_j^h} \rangle \frac{\partial a_{j,j'}^h}{\partial \boldsymbol{k}_t^h} && (14) \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_t^h, \partial_{\boldsymbol{y}_j^h} \rangle \frac{\partial}{\partial \boldsymbol{k}_t^h} \left( \frac{e^{\langle \boldsymbol{k}_t^h, \boldsymbol{q}_j^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_j^h \rangle}} \right) && (15) \\
&+ \sum_{j \leq T_{\text{aux}}} \sum_{j' \leq T_{\text{aux}}; j' \neq t} \langle \boldsymbol{v}_{j'}^h, \partial_{\boldsymbol{y}_j^h} \rangle \frac{\partial}{\partial \boldsymbol{k}_t^h} \left( \frac{e^{\langle \boldsymbol{k}_{j'}^h, \boldsymbol{q}_j^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_j^h \rangle}} \right) && (16) \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_t^h, \partial_{\boldsymbol{y}_j^h} \rangle \left[ \left( \frac{e^{\langle \boldsymbol{k}_t^h, \boldsymbol{q}_j^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_j^h \rangle}} \right) \boldsymbol{q}_j^h - \left( \frac{e^{\langle \boldsymbol{k}_t^h, \boldsymbol{q}_j^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_j^h \rangle}} \right)^2 \boldsymbol{q}_j^h \right] && (17) \\
&- \sum_{j \leq T_{\text{aux}}} \sum_{j' \leq T_{\text{aux}}; j' \neq t} \langle \boldsymbol{v}_{j'}^h, \partial_{\boldsymbol{y}_j^h} \rangle \left( \frac{e^{\langle \boldsymbol{k}_{j'}^h, \boldsymbol{q}_j^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_j^h \rangle}} \right) \left( \frac{e^{\langle \boldsymbol{k}_t^h, \boldsymbol{q}_j^h \rangle}}{\sum_{t' \leq T_{\text{aux}}} e^{\langle \boldsymbol{k}_{t'}^h, \boldsymbol{q}_j^h \rangle}} \right) \boldsymbol{q}_j^h && (18) \\
&= \sum_{j \leq T_{\text{aux}}} \langle \boldsymbol{v}_t^h, \partial_{\boldsymbol{y}_j^h} \rangle (a_{j,t}^h - (a_{j,t}^h)^2) \boldsymbol{q}_j^h - \sum_{j \leq T_{\text{aux}}} \sum_{j' \leq T_{\text{aux}}; j' \neq t} \langle \boldsymbol{v}_{j'}^h, \partial_{\boldsymbol{y}_j^h} \rangle a_{j,j'}^h a_{j,t}^h \boldsymbol{q}_j^h
\end{aligned}
$$

$$= \sum_{j \le T_{\text{aux}}} a^h_{j,t} \langle \partial_{\boldsymbol{y}^h_j}, \boldsymbol{v}^h_t - \sum_{j'} a^h_{j,j'} \boldsymbol{v}^h_{j'} \rangle \boldsymbol{q}^h_j$$

In Equation (14), we separate the inside sum into two components, since the derivative w.r.t. $\boldsymbol{k}^h_t$ differ for the two components, as outlined in the derivation of Equation (17) from Equation (15), and Equation (18) from Equation (16). We have skipped a step going from Equations (15) and (16) to Equations (17) and (18) due to typographical simplicity. The skipped step is extremely similar to Equation (12) in the derivation of $\partial_{\boldsymbol{q}^h_t}$. Finally, $\partial_{\boldsymbol{k}_t}$ is given by $\textsc{Vectorize}(\{\partial_{\boldsymbol{k}^h_t}\}_{h \le H_{\text{aux}}})$.

**Computation of $\partial_{\boldsymbol{v}_t}$:** Similar to the gradient computation of $\boldsymbol{q}_t$, the computation of $\partial_{\boldsymbol{v}_t}$ needs to be split across the $H_{\text{aux}}$ attention heads. However, like $\boldsymbol{k}_t$, $\boldsymbol{v}_t$ affects $\boldsymbol{y}_j$ for all $j \le T_{\text{aux}}$. For any head $h \le H_{\text{aux}}$, we follow the chain-rule step by step to get

$$\partial_{\boldsymbol{v}^h_t} = \sum_{j \le T_{\text{aux}}} (\frac{\partial \boldsymbol{y}^h_j}{\partial \boldsymbol{v}^h_t})^\top \partial_{\boldsymbol{y}^h_j} = \sum_{j \le T_{\text{aux}}} \left( \frac{\partial \sum_{j' \le T_{\text{aux}}} a_{j,j'} \boldsymbol{v}^h_{j'}}{\partial \boldsymbol{v}^h_t} \right)^\top \partial_{\boldsymbol{y}^h_j} = \sum_{j \le T_{\text{aux}}} a^h_{j,t} \partial_{\boldsymbol{y}^h_j}$$

□

**Theorem D.5.** *With the notations in Definitions D.1 to D.3, if on a given input sequence $\{\boldsymbol{x}_t\}_{t=1}^{T_{aux}}$, with its query, key, and value vectors $\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\}_{t=1}^{T_{aux}}$, all the $H_{aux}$ attention heads are $\varepsilon$-hard for some $\varepsilon > 0$, then for a given sequence of gradients $\{\partial_{\boldsymbol{y}_t}\}_{t=1}^{T_{aux}}$,*

$$\|\partial_{\boldsymbol{q}_t}\|_2, \|\partial_{\boldsymbol{k}_t}\|_2 \le \mathcal{O}(\varepsilon B_x^2 B_w^2 B_y), \quad \textit{for all } t \le T_{aux},$$

*where* $B_x = \max_{t \le T_{aux}} \|\boldsymbol{x}_t\|_2$, $B_y = \max_{t \le T_{aux}} \|\partial_{\boldsymbol{y}_t}\|_2$, *and* $B_w = \max\{\|\boldsymbol{W}_K\|_2, \|\boldsymbol{W}_Q\|_2, \|\boldsymbol{W}_V\|_2, \|\boldsymbol{b}_V\|_2, \|\boldsymbol{b}_K\|_2, \|\boldsymbol{b}_V\|_2\}$.

*This implies, for each position $t$,* $\left\| \widehat{\partial_{\boldsymbol{x}_t}} - \partial_{\boldsymbol{x}_t} \right\|_2 \le \mathcal{O}(\varepsilon B_x^2 B_w^3 B_y)$.

*Proof of Theorem D.5.* For typographical simplicity, we discuss the proof at an arbitrary position $t$. Recall the definition of an $\varepsilon$-hard attention head from Definition D.4. An attention head is defined to be $\varepsilon$-hard on an input sequence $\{\boldsymbol{x}_t\}_{t=1}^{T_{\text{aux}}}$, if for each position $t$, there exists a position $t_0$ such that the attention score $a_{t,t_0} \ge 1 - \varepsilon$.

For the proof, we simply focus on $\partial_{\boldsymbol{q}_t}$, and the proof for $\partial_{\boldsymbol{k}_t}$ follows like-wise.

**Bounds on $\boldsymbol{q}_t$:** Recalling the definition of $\partial_{\boldsymbol{q}_t}$ from Definition D.2, we have

$$\partial_{\boldsymbol{q}_t} = \textsc{Vectorize}(\{\sum_j a^h_{t,j} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_j)[\boldsymbol{k}^h_j - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}]\}_{h \le H_{\text{aux}}}).$$

Focusing on a head $h \le H_{\text{aux}}$, define $\partial_{\boldsymbol{q}^h_t} = \sum_j a^h_{t,j} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_j)[\boldsymbol{k}^h_j - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}]$ and $t_0 \le T_{\text{aux}}$ as the token position where the $\boldsymbol{q}_t$ attends the most to, i.e. $a^h_{t,t_0} \ge 1 - \varepsilon$ and $\sum_{j \le T_{\text{aux}}; j \ne t_0} a^h_{t,j} \le \varepsilon$. Then,

$$\begin{aligned}
\left\| \partial_{\boldsymbol{q}^h_t} \right\|_2 &= \left\| \sum_j a^h_{t,j} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_j)[\boldsymbol{k}^h_j - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}] \right\|_2 \\
&= \left\| a^h_{t,t_0} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_{t_0})[\boldsymbol{k}^h_{t_0} - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}] + \sum_{j \ne t_0} a^h_{t,j} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_j)[\boldsymbol{k}^h_j - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}] \right\|_2 \\
&\le \underbrace{\left\| a^h_{t,t_0} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_{t_0})[\boldsymbol{k}^h_{t_0} - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}] \right\|_2}_{\text{Term1}} + \underbrace{\left\| \sum_{j \ne t_0} a^h_{t,j} ((\partial_{\boldsymbol{y}^h_t})^\top \boldsymbol{v}^h_j)[\boldsymbol{k}^h_j - \sum_{j'} a^h_{t,j'} \boldsymbol{k}^h_{j'}] \right\|_2}_{\text{Term2}},
\end{aligned}$$

where the final step uses a Cauchy-Schwartz inequality. We focus on the two terms separately.

1. Term1: Focusing on $\boldsymbol{k}_{t_0}^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h$, we have

$$\begin{aligned}\left\|\boldsymbol{k}_{t_0}^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h\right\|_2 &= \left\|(1-a_{t,t_0})\boldsymbol{k}_{t_0}^h - \sum_{j'\neq t_0} a_{t,j'}^h \boldsymbol{k}_{j'}^h\right\|_2 \\ &\leq (1-a_{t,t_0})\left\|\boldsymbol{k}_{t_0}^h\right\|_2 + \sum_{j'\neq t_0} a_{t,j'}^h \left\|\boldsymbol{k}_{j'}^h\right\|_2 \\ &\leq ((1-a_{t,t_0}) + \sum_{j'\neq t_0} a_{t,j'}^h) \max_j \left\|\boldsymbol{k}_j^h\right\|_2 \\ &\leq 2\varepsilon \max_j \left\|\boldsymbol{k}_j^h\right\|_2 . \end{aligned} \tag{19}$$

   We use a Cauchy-Schwartz inequality in the second and third steps and the attention head behavior in the final step. Hence, Term1 can now be bounded as follows:

$$\begin{aligned}\left\|a_{t,t_0}^h((\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_{t_0}^h)[\boldsymbol{k}_{t_0}^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h]\right\|_2 &= a_{t,t_0}^h \left|(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_{t_0}^h\right| \left\|\boldsymbol{k}_{t_0}^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h\right\|_2 , \\ &\leq 2\varepsilon \left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \left\|\boldsymbol{v}_{t_0}^h\right\|_2 \max_j \left\|\boldsymbol{k}_j^h\right\|_2 .\end{aligned}$$

   In the final step, in addition to the bound from Equation (19), we use a Cauchy-Schwartz inequality to bound $\left|(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_{t_0}^h\right|$ and bound the attention score $a_{t,t_0}^h$ by 1.

2. Term2: Focusing on $\boldsymbol{k}_j^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h$ for any $j \leq T_{\text{aux}}$, we have using two Cauchy-Schwartz inequalities:

$$\left\|\boldsymbol{k}_j^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h\right\|_2 \leq \left\|\boldsymbol{k}_j^h\right\|_2 + \left\|\sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h\right\|_2 \leq (1 + \sum_{j'} a_{t,j'}^h) \max_{j'} \left\|\boldsymbol{k}_{j'}^h\right\|_2 = 2 \max_{j'} \left\|\boldsymbol{k}_{j'}^h\right\|_2 . \tag{20}$$

   Hence,

$$\begin{aligned}\left\|\sum_{j\neq t_0} a_{t,j}^h((\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h)[\boldsymbol{k}_j^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h]\right\|_2 &\leq \left(\sum_{j\neq t_0} a_{t,j}^h\right) \max_j \left|(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\right| \left\|\boldsymbol{k}_j^h - \sum_{j'} a_{t,j'}^h \boldsymbol{k}_{j'}^h\right\|_2 \\ &\leq 2\varepsilon \left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \left(\max_j \left\|\boldsymbol{v}_j^h\right\|_2\right) \left(\max_{j'} \left\|\boldsymbol{k}_{j'}^h\right\|_2\right) .\end{aligned}$$

   In the final step, in addition to the bound from Equation (20), we use a Cauchy-Schwartz inequality to bound $\left|(\partial_{\boldsymbol{y}_t^h})^\top \boldsymbol{v}_j^h\right|$ and use the $\varepsilon$-hard behavior of the attention head to bound $\sum_{j\neq t_0} a_{t,j}^h$.

Combining the bounds on both terms, we have

$$\begin{aligned}\left\|\partial_{\boldsymbol{q}_t^h}\right\|_2 &\leq 2\varepsilon \left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \left\|\boldsymbol{v}_{t_0}^h\right\|_2 \max_j \left\|\boldsymbol{k}_j^h\right\|_2 + 2\varepsilon \left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \left(\max_j \left\|\boldsymbol{v}_j^h\right\|_2\right) \left(\max_{j'} \left\|\boldsymbol{k}_{j'}^h\right\|_2\right) \\ &\leq 4\varepsilon \left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \left(\max_j \left\|\boldsymbol{v}_j^h\right\|_2\right) \left(\max_{j'} \left\|\boldsymbol{k}_{j'}^h\right\|_2\right) .\end{aligned}$$

We bound the remaining terms as follows.

- $\left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \leq B_y$, under the bounded assumption of the gradients.
- For any $j \leq T_{\text{aux}}$, we have $\left\|\boldsymbol{k}_j^h\right\|_2 \leq \|\boldsymbol{k}_j\|_2$ since $\boldsymbol{k}_j = \text{VECTORIZE}(\{\boldsymbol{k}_j^{h'}\}_{h'\in H_{\text{aux}}})$. Furthermore, from the defintion of the key vector $\boldsymbol{k}_j$, $\|\boldsymbol{k}_j\|_2 = \|\boldsymbol{W}_K \boldsymbol{x}_j + \boldsymbol{b}_K\|_2 \leq \|\boldsymbol{W}_K\|_2 \|\boldsymbol{x}_j\|_2 + \|\boldsymbol{b}_K\|_2$ with a Cauchy-Schwartz inequality. Under the bounded assumptions of $\boldsymbol{W}_K, \boldsymbol{b}_K$ and input $\boldsymbol{x}_j$, we have $\|\boldsymbol{k}_j\|_2 \leq B_w(1+B_x)$.

- Similar procedure can be followed for bounding $\max_j \left\|\boldsymbol{v}_j^h\right\|_2$.

Thus, we have $\left\|\partial_{\boldsymbol{q}_t^h}\right\|_2 \leq 4\varepsilon \left\|\partial_{\boldsymbol{y}_t^h}\right\|_2 \left(\max_j \left\|\boldsymbol{v}_j^h\right\|_2\right) \left(\max_{j'} \left\|\boldsymbol{k}_{j'}^h\right\|_2\right) \leq 4\varepsilon B_w^2 (1+B_x)^2 B_y$.

**Bounds on $\left\|\widehat{\partial_{\boldsymbol{x}_t}} - \partial_{\boldsymbol{x}_t}\right\|_2$:** From the definitons of $\widehat{\partial_{\boldsymbol{x}_t}}$ and $\partial_{\boldsymbol{x}_t}$ from Definition D.3, we have

$$\begin{aligned}\left\|\widehat{\partial_{\boldsymbol{x}_t}} - \partial_{\boldsymbol{x}_t}\right\|_2 &= \left\|\boldsymbol{W}_K^\top \partial_{\boldsymbol{k}_t} + \boldsymbol{W}_Q^\top \partial_{\boldsymbol{q}_t}\right\|_2 \leq \left\|\boldsymbol{W}_K\right\|_2 \left\|\partial_{\boldsymbol{k}_t}\right\|_2 + \left\|\boldsymbol{W}_Q\right\|_2 \left\|\partial_{\boldsymbol{q}_t}\right\|_2 \\ &\leq 8\varepsilon B_w^3 (1+B_x)^2 B_y = \mathcal{O}(\varepsilon B_w^3 B_x^2 B_y),\end{aligned}$$

where we use Cauchy-schwartz inequality in the second step. We use the assumed bounds on $\left\|\boldsymbol{W}_Q\right\|_2, \left\|\boldsymbol{W}_K\right\|_2$, and the computed bounds on $\left\|\partial_{\boldsymbol{q}_t}\right\|_2, \left\|\partial_{\boldsymbol{k}_t}\right\|_2$ in the pre-final step. □

## E. Layer normalization

**Definition E.1.** [Layer Normalization] Define a normalization function $f : \mathbb{R}^d \to \mathbb{R}^d$ that performs $f(\boldsymbol{x}) = (\boldsymbol{x} - \mu)/\sigma$, where $\mu$ and $\sigma$ are the mean and standard deviation of $\boldsymbol{x}$, respectively. Then, layer normalization with parameters $\gamma, \boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$ takes as input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ and outputs $\boldsymbol{y} \in \mathbb{R}^{D_{\text{aux}}}$, which is computed as $\boldsymbol{z} = f(\boldsymbol{x}), \boldsymbol{y} = \gamma \odot \boldsymbol{z} + \boldsymbol{b}$.

**Definition E.2.** [Exact Gradient for Layer Normalization] Using notations in Definition E.1, given the gradient of the loss w.r.t the output of the Layer Normalization $\partial_{\boldsymbol{y}}$, backpropagation computes $\partial_{\boldsymbol{x}}$ as

$$\partial_{\boldsymbol{x}} = (\partial_{\boldsymbol{z}} - {D_{\text{aux}}}^{-1} \sum_{i=1}^{D_{\text{aux}}} \partial_{z_i} - \langle \partial_{\boldsymbol{z}}, \boldsymbol{z} \rangle \boldsymbol{z})/\sigma \qquad \partial_{\boldsymbol{z}} = \gamma \odot \partial_{\boldsymbol{y}}.$$

Exact backpropagation is expensive because $\langle \partial_{\boldsymbol{z}}, \boldsymbol{z} \rangle \boldsymbol{z}$ requires using at least two sequential MLPs. We thus approximate it with a first-order Taylor expansion, which is entry-wise close to the true gradient.

**Definition E.3.** [$\epsilon$-approximate Layer Normalization Gradient] With notations defined above, this layer takes $\partial_{\boldsymbol{y}}, \boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\widehat{\partial_{\boldsymbol{x}}} = \frac{1}{\epsilon}(f(\boldsymbol{x} + \epsilon\gamma \odot \partial_{\boldsymbol{y}}) - f(\boldsymbol{x}))$.

In the discussions below, we consider a layer normalization layer in the auxiliary model with parameters $\{\gamma, \boldsymbol{b}\}$ that takes in input sequence $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T_{\text{aux}}}$ and outputs $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{T_{\text{aux}}}$, with $\boldsymbol{y}_t = \gamma \odot \boldsymbol{z}_t + \boldsymbol{b}; \boldsymbol{z}_t = f(\boldsymbol{x}_t)$ for each $t \leq T_{\text{aux}}$. Since this involves a token-wise operation, we will present our constructed modules with a general token position $t$ and the prefix tokens $\{\boldsymbol{v}_j\}$. We will use $\boldsymbol{W}_\gamma$ as a diagonal matrix in $\mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$, containing $\gamma$ on its main diagonal.

**TINT Layer normalization Forward module** The input embedding to this module $\boldsymbol{e}_t$ will contain $\boldsymbol{x}_t$ in its first $D_{\text{aux}}$ coordinates. The layer normalization computation can be divided into two sub-operations: (a) application of $f$, and (b) linear computation using $\gamma, \boldsymbol{b}$. We will present a TINT module for each sub-operation.

We can represent the function $f$ using a layer normalization operation itself, with its weight and bias parameters set as $\mathbf{1}$ and $\mathbf{0}$ respectively. However, since the relevant input exists only in the first $D_{\text{aux}}$ coordinates, the operation on the first $D_{\text{aux}}$ coordinates needs to be independent of the rest of the coordinates. To do so, we instead use Group normalization (Definition E.6) on $\boldsymbol{e}_t$, with groups of size $D_{\text{aux}}$.

Now, the embedding $\boldsymbol{e}_t$ contains $f(\boldsymbol{x}_t)$ in its first $D_{\text{aux}}$ coordinates. The second sub-operation can then be viewed as a Linear Layer computation, i.e. $\boldsymbol{y}_t = \boldsymbol{W}_\gamma \boldsymbol{x}_t + \boldsymbol{b}$. Hence, we simply stack the rows of $\boldsymbol{W}_\gamma$ and $\boldsymbol{b}_\gamma$ onto the prefix tokens $\{\boldsymbol{v}_j\}$ and call the TINT Linear Forward module (Appendix C).

**Auxiliary's gradient backpropagation through layer normalization** With the definition of layer normalization and the normalization function $f$ in Definition E.1, the auxiliary's backpropagation operation takes in the loss gradient w.r.t. output ($\partial_{\boldsymbol{y}}$) and computes the loss gradient w.r.t. input ($\partial_{\boldsymbol{x}}$).

**Definition E.2.** [Exact Gradient for Layer Normalization] Using notations in Definition E.1, given the gradient of the loss w.r.t the output of the Layer Normalization $\partial_{\boldsymbol{y}}$, backpropagation computes $\partial_{\boldsymbol{x}}$ as

$$\partial_{\boldsymbol{x}} = (\partial_{\boldsymbol{z}} - {D_{\text{aux}}}^{-1} \sum_{i=1}^{D_{\text{aux}}} \partial_{z_i} - \langle \partial_{\boldsymbol{z}}, \boldsymbol{z} \rangle \boldsymbol{z})/\sigma \qquad \partial_{\boldsymbol{z}} = \gamma \odot \partial_{\boldsymbol{y}}.$$

**Complexity of true backpropagation** The above operation is computation heavy since it involves computing (a) $\partial_{\boldsymbol{z}}$, (b) $f(\partial_{\boldsymbol{z}})$, (c) $\langle \partial_{\boldsymbol{z}}, \boldsymbol{z} \rangle \boldsymbol{z}$, and (d) multiplying by a factor of $\frac{1}{\sigma}$. $\langle \partial_{\boldsymbol{z}}, \boldsymbol{z} \rangle \boldsymbol{z}$ in itself will require two MLP layers, following Lemma B.5. In order to reduce the number of layers, we turn to first-order Taylor expansion for approximating the above operation.

**Definition E.3.** [$\epsilon$-approximate Layer Normalization Gradient] With notations defined above, this layer takes $\partial_{\boldsymbol{y}}, \boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\widehat{\partial_{\boldsymbol{x}}} = \frac{1}{\epsilon}(f(\boldsymbol{x} + \epsilon\gamma \odot \partial_{\boldsymbol{y}}) - f(\boldsymbol{x}))$.

The following theorem shows that the first-order gradient is a good approximation of the true gradient, and in the limit of $\epsilon$ tending to 0, the approximation error tends to 0 as well.

**Theorem E.4.** *For any $\epsilon > 0$, and a layer normalization layer with parameters $\gamma, \boldsymbol{b} \in \mathbb{R}^{D_{aux}}$, for an input $\boldsymbol{x} \in \mathbb{R}^{D_{aux}}$ and gradient $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{aux}}$,*

$$\left\| \widehat{\partial_{\boldsymbol{x}}} - \partial_{\boldsymbol{x}} \right\|_2 \leq \mathcal{O}(\epsilon D_{aux}^{3/2} \sigma^{-2} \left\|\gamma\right\|_2^2 \left\|\partial_{\boldsymbol{y}}\right\|_2^2),$$

*where $\sigma$ denotes the standard deviation of $\boldsymbol{x}$. $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ have been computed from $\boldsymbol{x}$, $\partial_{\boldsymbol{y}}$ and $\epsilon$ using Definitions E.2 and E.3.*

**TINT Layer normalization backpropagation module** The input embeddings $\boldsymbol{e}_t$ contain $\partial_{\boldsymbol{y}_t}$ at each position $t$ in the first $D_{\text{aux}}$ coordinates. Since we further need the input to the auxiliary's layer normalization layer under consideration, we copy $\boldsymbol{x}_t$ from the TINT Layer normalization Forward module at each position $t$ using residual connections. Furthermore, residual connections have been used to copy the contents of the prefix tokens $\{\boldsymbol{v}_j\}$ from the Layer normalization Forward module, which contain $\boldsymbol{W}_\gamma, \boldsymbol{b}$. Recall that for ease of presentation, we use $\boldsymbol{z}_t$ to represent $f(\boldsymbol{x}_t)$.

We set $\epsilon$ as a hyperparameter and return $\widehat{\partial_{\boldsymbol{x}}}$ as the output of this module. The computation of $\widehat{\partial_{\boldsymbol{x}}}$ can be divided into two sub-operations: (a) computation of $\partial_{\boldsymbol{z}_t} := \gamma \odot \partial_{\boldsymbol{y}_t}$, and (b) computation of $\frac{1}{\epsilon}(f(\boldsymbol{x}_t + \epsilon\partial_{\boldsymbol{z}_t}) - f(\boldsymbol{x}_t))$. We represent each sub-operation as a TINT module.

To compute $\partial_{\boldsymbol{z}_t} := \gamma \odot \partial_{\boldsymbol{y}_t} = W_\gamma \partial_{\boldsymbol{y}_t}$, we can observe that the required operation is identical to backpropagating through a linear layer with parameters $\boldsymbol{W}_\gamma$ and $\boldsymbol{b}$. Hence, we simply call the Linear Backpropagation module on the current embeddings. We use residual connections to retain $\boldsymbol{x}_t$ at each location $t$, and the contents of the prefix tokens $\{\boldsymbol{v}_j\}$.

Now, the embedding $\boldsymbol{e}_t$ contains $\partial_{\boldsymbol{z}_t}$ and $\boldsymbol{x}_t$. In order to backpropagate through $f$, we first use a linear layer to compute $\boldsymbol{x}_t + \epsilon\partial_{\boldsymbol{z}_t}$ and retain $\boldsymbol{x}_t$. Following the same procedure as the Forward module, we use a Group normalization layer with weight and bias parameters $\mathbf{1}$ and $\mathbf{0}$ respectively, to compute $f(\boldsymbol{x}_t + \epsilon\partial_{\boldsymbol{z}_t})$ and $f(\boldsymbol{x}_t)$. Finally, we use a linear layer to compute $\frac{1}{\epsilon}(f(\boldsymbol{x}_t + \epsilon\partial_{\boldsymbol{z}_t}) - f(\boldsymbol{x}_t))$.

**Auxiliary's Descent update** And finally, the auxiliary's descent operation updates parameters $\gamma, \boldsymbol{b}$ using a batch of inputs $\{\boldsymbol{x}_t\}_{t \leq T}$ and the loss gradient w.r.t. the corresponding outputs $\{\partial_{\boldsymbol{y}_t}\}_{t \leq T}$.

**Definition E.5** (Auxiliary's layer normalization descent)**.** For parameters $\gamma, \boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$, descent update takes in a batch of inputs $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t \leq T_{\text{aux}}}$ and gradients $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^{D_{\text{aux}}}\}_{t \leq T_{\text{aux}}}$ and updates the parameters as follows:

$$\gamma \leftarrow \gamma - \eta \sum_{t \leq T_{\text{aux}}} \partial_{\boldsymbol{y}_t} \odot \boldsymbol{z}_t; \qquad \boldsymbol{b} \leftarrow \boldsymbol{b} - \eta \sum_{t \leq T_{\text{aux}}} \partial_{\boldsymbol{y}_t},$$

where $\boldsymbol{z}_t$ represents $f(\boldsymbol{x}_t)$.

The update of $\gamma$ involves an elementwise multiplication between $\partial_{\boldsymbol{y}_t}$ and $\boldsymbol{z}_t$, which requires an MLP layer (Lemma B.5). With the prefix tokens containing the rows of $\boldsymbol{W}_\gamma$ and $\boldsymbol{b}$, we instead consider the update of $\boldsymbol{b}$ alone with the descent update.

**TINT Layer normalization descent module** The input embeddings contain $\partial_{\boldsymbol{y}_t}$ in the first $D_{\text{aux}}$ coordinates. The prefix tokens contain $\boldsymbol{W}_\gamma, \boldsymbol{b}$, which have been copied from the Forward module using residual connections. The update of $\boldsymbol{b}$ is identical to the auxiliary's descent update through a linear layer. Hence, we apply a TINT Linear descent module to the current embeddings, updating only the bias $\boldsymbol{b}$ and switching off the update to $\boldsymbol{W}_\gamma$.

### E.1. Additional definitions

We describe TINT group normalization layer below, which we use in different modules to simulate the auxiliary's layer normalization operations.

**Definition E.6** (TINT $D_{\text{aux}}$-Group normalization)**.** Define a normalization function $f : \mathbb{R}^d \to \mathbb{R}^d$ that performs $f(\boldsymbol{x}) = (\boldsymbol{x} - \mu)/\sigma$, where $\mu$ and $\sigma$ are the mean and standard deviation of $\boldsymbol{x}$, respectively. Then, $D_{\text{aux}}$-Group RMSnorm with parameters $\gamma^{\text{TINT}}, \boldsymbol{b}^{\text{TINT}} \in \mathbb{R}^{D_{\text{aux}}}$ takes as input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{sim}}}$ and outputs $\boldsymbol{y} = \text{VECTORIZE}(\{\boldsymbol{y}^h \in \mathbb{R}^{D_{\text{aux}}}\}_{h \le \lfloor D_{\text{sim}}/D_{\text{aux}} \rfloor})$, with

$$\boldsymbol{y}^h = \gamma^{\text{TINT}} \odot f(\boldsymbol{x}^h) + \boldsymbol{b}^{\text{TINT}},$$

where $\boldsymbol{x}^h = \text{SPLIT}_{\lfloor D_{\text{sim}}/D_{\text{aux}} \rfloor}(\boldsymbol{x})_h$.

### E.2. Proof of theorems and gradient definitions

We restate the theorems and definitions, before presenting their proofs for easy referencing.

**Definition E.2.** [Exact Gradient for Layer Normalization] Using notations in Definition E.1, given the gradient of the loss w.r.t the output of the Layer Normalization $\partial_{\boldsymbol{y}}$, backpropagation computes $\partial_{\boldsymbol{x}}$ as

$$\partial_{\boldsymbol{x}} = (\partial_{\boldsymbol{z}} - {D_{\text{aux}}}^{-1} \sum_{i=1}^{D_{\text{aux}}} \partial_{z_i} - \langle \partial_{\boldsymbol{z}}, \boldsymbol{z} \rangle \boldsymbol{z})/\sigma \qquad \partial_{\boldsymbol{z}} = \gamma \odot \partial_{\boldsymbol{y}}.$$

*Derivation of gradient in Definition E.2* . With the normalization function $f$ and parameters $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$, recall from Definition E.1 that given an input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$, a layer normalization layer returns $\boldsymbol{y} = \gamma \odot \boldsymbol{z} + \boldsymbol{b}; \boldsymbol{z} = f(\boldsymbol{x})$. Let $\mu$ and $\sigma$ denote the mean and standard deviation of $\boldsymbol{x}$. They can be computed as

$$\mu = \frac{1}{D_{\text{aux}}} \sum_{i=1}^{D_{\text{aux}}} x_i, \quad \sigma = \sqrt{\frac{1}{D_{\text{aux}}} \sum_{i=1}^{D_{\text{aux}}} (x_i - \mu)^2}.$$

With the chain rule, we can compute $\partial_{\boldsymbol{x}}$ from $\partial_{\boldsymbol{y}}$ as follows.

$$\partial_{\boldsymbol{x}} = (\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})^\top \partial_{\boldsymbol{z}}; \quad \text{with } \partial_{\boldsymbol{z}} = (\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}})^\top \partial_{\boldsymbol{y}}. \tag{21}$$

Since $\boldsymbol{y} = \gamma \odot \boldsymbol{z} + \boldsymbol{b}$, we have $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} = \boldsymbol{W}_\gamma$, where $\boldsymbol{W}_\gamma$ represents a diagonal matrix with $\gamma$ on the main diagonal. Thus, $\partial_{\boldsymbol{z}} = \boldsymbol{W}_\gamma \partial_{\boldsymbol{y}} = \gamma \odot \partial_{\boldsymbol{y}}$.

With $\boldsymbol{z} = f(\boldsymbol{x}) = \frac{\boldsymbol{x} - \mu}{\sigma}$, we have

$$\begin{aligned} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} &= \frac{\partial}{\partial \boldsymbol{x}} \left( \frac{\boldsymbol{x} - \mu}{\sigma} \right) = \frac{1}{\sigma} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{x}} - \frac{1}{\sigma} \frac{\partial \mu}{\partial \boldsymbol{x}} - \frac{(\boldsymbol{x} - \mu)}{\sigma^2} \left( \frac{\partial \sigma}{\partial \boldsymbol{x}} \right)^\top \\ &= \frac{1}{\sigma} \left( \boldsymbol{I} - \frac{1}{D_{\text{aux}}} \boldsymbol{1}\boldsymbol{1}^\top - \boldsymbol{z}\boldsymbol{z}^\top \right). \end{aligned} \tag{22}$$

In the final step, we require $\frac{\partial \mu}{\partial \boldsymbol{x}}$ and $\frac{\partial \sigma}{\partial \boldsymbol{x}}$, which are computed as follows.

- $\frac{\partial \mu}{\partial \boldsymbol{x}} \in \mathbb{R}^{D_{\text{aux}}}$ with its $j$th element given by

$$\left( \frac{\partial \mu}{\partial \boldsymbol{x}} \right)_j = \frac{\partial \mu}{\partial x_j} = \frac{\partial}{\partial x_j} (\frac{1}{D_{\text{aux}}} \sum_{i=1}^{D_{\text{aux}}} x_i) = \frac{1}{D_{\text{aux}}}.$$

- $\frac{\partial \sigma}{\partial \boldsymbol{x}} \in \mathbb{R}^{D_{\text{aux}}}$ with its $j$th element given by

$$\begin{aligned}\left(\frac{\partial \sigma}{\partial \boldsymbol{x}}\right)_j = \frac{\partial \sigma}{\partial x_j} &= \frac{\partial}{\partial x_j}\left(\sqrt{\frac{1}{D_{\text{aux}}}\sum_{i=1}^{D_{\text{aux}}}(x_i-\mu)^2}\right) \\ &= \frac{1}{\sqrt{\sum_{i=1}^{D_{\text{aux}}}(x_i-\mu)^2}}\sum_{i=1}^{D_{\text{aux}}}(x_i-\mu)\frac{\partial(x_i-\mu)}{\partial x_j} \\ &= \frac{1}{\sqrt{\sum_{i=1}^{D_{\text{aux}}}(x_i-\mu)^2}}\left((x_j-\mu)-\frac{1}{D_{\text{aux}}}\sum_{i=1}^{D_{\text{aux}}}(x_i-\mu)\right) = \frac{x_j-\mu}{\sigma} := z_j,\end{aligned}$$

  where we have re-utilized the $\frac{\partial \mu}{\partial \boldsymbol{x}}$ in the pre-final step.

Hence, from Equation (21),

$$\partial_{\boldsymbol{x}} = (\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})^\top \partial_{\boldsymbol{z}} = \frac{1}{\sigma}\left(\boldsymbol{I} - \frac{1}{D_{\text{aux}}}\mathbf{1}\mathbf{1}^\top - \boldsymbol{z}\boldsymbol{z}^\top\right)\partial_{\boldsymbol{z}} = \frac{1}{\sigma}\left(\partial_{\boldsymbol{z}} - \frac{1}{D_{\text{aux}}}\langle \mathbf{1}, \partial_{\boldsymbol{z}}\rangle \mathbf{1} - \langle \boldsymbol{z}, \partial_{\boldsymbol{z}}\rangle \boldsymbol{z}\right).$$

□

We repeat Theorem E.4 for easier reference.

**Theorem E.4.** *For any $\epsilon > 0$, and a layer normalization layer with parameters $\gamma, \boldsymbol{b} \in \mathbb{R}^{D_{aux}}$, for an input $\boldsymbol{x} \in \mathbb{R}^{D_{aux}}$ and gradient $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{aux}}$,*

$$\left\|\widehat{\partial_{\boldsymbol{x}}} - \partial_{\boldsymbol{x}}\right\|_2 \leq \mathcal{O}(\epsilon D_{aux}^{3/2}\sigma^{-2}\left\|\gamma\right\|_2^2\left\|\partial_{\boldsymbol{y}}\right\|_2^2),$$

*where $\sigma$ denotes the standard deviation of $\boldsymbol{x}$. $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ have been computed from $\boldsymbol{x}$, $\partial_{\boldsymbol{y}}$ and $\epsilon$ using Definitions E.2 and E.3.*

*Proof of Theorem E.4* . With the normalization function $f$ and parameters $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$, recall from Definition E.1 that given an input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$, a layer normalization layer returns $\boldsymbol{y} = \gamma \odot \boldsymbol{z} + \boldsymbol{b}; \boldsymbol{z} = f(\boldsymbol{x})$. Let $\mu$ and $\sigma$ denote the mean and standard deviation of $\boldsymbol{x}$. They can be computed as

$$\mu = \frac{1}{D_{\text{aux}}}\sum_{i=1}^{D_{\text{aux}}} x_i, \quad \sigma = \sqrt{\frac{1}{D_{\text{aux}}}\sum_{i=1}^{D_{\text{aux}}}(x_i-\mu)^2}.$$

We will refer to $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$ from Equation (22) and the formulation of $\partial_{\boldsymbol{x}}$ from Equation (21) for our current proof. To recall, they are

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{1}{\sigma}\left(\boldsymbol{I} - \frac{1}{D_{\text{aux}}}\mathbf{1}\mathbf{1}^\top - \boldsymbol{z}\boldsymbol{z}^\top\right), \qquad \partial_{\boldsymbol{x}} = (\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}})^\top \partial_{\boldsymbol{z}}.$$

Using a second-order Taylor expansion of the normalization function $f$ around $\boldsymbol{x}$, we have

$$\begin{aligned}f(\boldsymbol{x}+\epsilon\partial_{\boldsymbol{z}}) &= f(\boldsymbol{x}) + \epsilon\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\partial_{\boldsymbol{z}} + \int_0^\epsilon \partial_{\boldsymbol{z}}^\top \frac{\partial}{\partial \boldsymbol{x}_\theta}\left(\frac{\partial f(\boldsymbol{x}_\theta)}{\partial \boldsymbol{x}_\theta}\right)\partial_{\boldsymbol{z}}\theta d\theta \\ &= f(\boldsymbol{x}) + \epsilon\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\partial_{\boldsymbol{z}} - \int_0^\epsilon \frac{1}{\sigma_\theta^2}\left(\left\|\partial_{\boldsymbol{z}}\right\|_2^2 - \frac{1}{D_{\text{aux}}}\sum_{i=1}^{D_{\text{aux}}}(\langle \mathbf{1}, \partial_{\boldsymbol{z}}\rangle)^2 - (\langle \boldsymbol{z}_\theta, \partial_{\boldsymbol{z}}\rangle)^2 \boldsymbol{z}_\theta\right)\theta d\theta,\end{aligned}$$

where $\boldsymbol{x}_\theta$ represents $\boldsymbol{x} + \theta\partial_{\boldsymbol{z}}, \boldsymbol{z}_\theta = f(\boldsymbol{x}_\theta)$. The second step follows similar steps for computing $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$ in Equation (22). We avoid this computation since we only need to make sure that the second-order term is bounded. Furthermore, if

$\epsilon \leq \mathcal{O}\left(\frac{\sigma}{\sqrt{D_{\text{aux}}}\|\partial_{\boldsymbol{z}}\|_2}\right)$, we can show the $\ell_2$-norm of the second-order term can be bounded by $\mathcal{O}(\epsilon^2 D_{\text{aux}}^{3/2}\sigma^{-2}\|\partial_{\boldsymbol{z}}\|_2^2)$. We avoid this computation as well.

Thus, from the above formulation, we have

$$\lim_{\epsilon\to 0}\frac{f(\boldsymbol{x}+\epsilon\partial_{\boldsymbol{z}})-f(\boldsymbol{x})}{\epsilon} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\partial_{\boldsymbol{z}} = \left(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^\top \partial_{\boldsymbol{z}} = \partial_{\boldsymbol{x}}.$$

The pre-final step follows from Equation (22), where $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{1}{\sigma}\left(\boldsymbol{I} - \frac{1}{D_{\text{aux}}}\boldsymbol{1}\boldsymbol{1}^\top - \boldsymbol{z}\boldsymbol{z}^\top\right)$ can be shown to be symmetric. The final step follows from the gradient formulation in Equation (21). Including the error term, we have the final bound as

$$\left\|\frac{f(\boldsymbol{x}+\epsilon\partial_{\boldsymbol{z}})-f(\boldsymbol{x})}{\epsilon} - \partial_{\boldsymbol{x}}\right\|_2 \leq \mathcal{O}(\epsilon D_{\text{aux}}^{3/2}\sigma^{-2}\|\partial_{\boldsymbol{z}}\|_2^2).$$

Using $\partial_{\boldsymbol{z}} = \gamma \odot \partial_{\boldsymbol{y}}$ and a Cauchy-Schwartz inequality gives the final bound. □

## F. Activation layer

**Definition F.1** (Auxiliary activation)**.** For a continuous function $\sigma_{\text{act}} : \mathbb{R} \to \mathbb{R}$, an activation layer takes $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\boldsymbol{y} = \sigma_{\text{act}}(\boldsymbol{x})$ with $y_i = \sigma_{\text{act}}(x_i)$ for all $i \leq D_{\text{aux}}$.

In the discussions below, we consider an activation layer in the auxiliary model with activation function $\sigma_{\text{act}}$ that takes in input sequence $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T_{\text{aux}}}$ and outputs $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{T_{\text{aux}}}$, with $\boldsymbol{y}_t = \sigma_{\text{act}}(\boldsymbol{x}_t)$ for each $t \leq T_{\text{aux}}$. Since this involves a token-wise operation, we will present our constructed modules with a general token position $t$. Since no parameters of the auxiliary model are involved in this operation, the prefix tokens $\{\boldsymbol{v}_j\}$ contain 0 in the following modules.

**TINT Activation Forward module** The embedding $\boldsymbol{e}_t$ contains $\boldsymbol{x}_t$ in its first $D_{\text{aux}}$ indices. We simply pass the embeddings into activation $\sigma_{\text{act}}$, which returns $\sigma_{\text{act}}(\boldsymbol{x}_t)$ in its first $D_{\text{aux}}$ indices.

**Auxiliary's backpropagation through activation** With the definition in Definition F.1, the auxiliary's backpropagation takes in the loss gradient w.r.t. output ($\partial_{\boldsymbol{y}}$) and computes the loss gradient w.r.t. input ($\partial_{\boldsymbol{x}}$). We further assume that the derivative of $\sigma_{\text{act}}$ is well-defined everywhere. This assumption includes non-differentiable activation functions with well-defined derivatives like $ReLU$.

**Definition F.2** (Auxiliary activation backpropagation)**.** For a continuous function $\sigma_{\text{act}} : \mathbb{R} \to \mathbb{R}$, with a well-defined derivative $\sigma'_{\text{act}}(x) = \partial\sigma_{\text{act}}(x)/\partial x$ for each $x \in \mathbb{R}$, the backpropagation takes $\partial_{\boldsymbol{y}}, \boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs

$$\partial_{\boldsymbol{x}} = \sigma'_{\text{act}}(\boldsymbol{x}) \odot \partial_{\boldsymbol{y}},$$

where $\sigma'_{\text{act}}(\boldsymbol{x}) \in \mathbb{R}^{D_{\text{aux}}}$ with $\sigma'_{\text{act}}(\boldsymbol{x})_i = \sigma'_{\text{act}}(x_i)$ at each $i \leq D_{\text{aux}}$.

**Complexity of true backpropagation** The above operation is computation heavy since it involves $\sigma'_{\text{act}}(\boldsymbol{x}) \odot \partial_{\boldsymbol{y}}$. As mentioned for the layer normalization module, the element-wise multiplication between $\sigma'_{\text{act}}(\boldsymbol{x})$ and $\partial_{\boldsymbol{y}}$ will require an MLP module following Lemma B.5. Furthermore, it involves changing the activation function in TINT in specific modules to $\sigma'_{\text{act}}$. To circumvent this, we instead turn to a first-order Taylor approximation.

**Definition F.3** (Approximate Activation backpropagation)**.** For a continuous function $\sigma_{\text{act}} : \mathbb{R} \to \mathbb{R}$ and a hyperparameter $\epsilon$, the layer takes $\partial_{\boldsymbol{y}}, \boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs

$$\widehat{\partial_{\boldsymbol{x}}} = \frac{1}{\epsilon}\left(\sigma_{\text{act}}(\boldsymbol{x}+\epsilon\partial_{\boldsymbol{y}}) - \sigma_{\text{act}}(\boldsymbol{x})\right).$$

The following theorems show that under mild assumptions on the activation function and the input, gradient pair, the first-order gradient is a good approximation to the true gradient.

**Theorem F.4.** *For any $\epsilon > 0$, $B_y, B_{act} > 0$, consider a second-order differentiable activation function $\sigma_{act} : \mathbb{R} \to \mathbb{R}$, with $\partial^2 \sigma_{act}(x)/\partial(x^2)$ bounded by $B_{act}$ for each $x \in \mathbb{R}$. Then, for any input $\boldsymbol{x} \in \mathbb{R}^{D_{aux}}$ and gradient $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{aux}}$ with $\|\partial_{\boldsymbol{y}}\|_2 \leq B_y$, the following holds true:*

$$\left\|\partial_{\boldsymbol{x}} - \widehat{\partial_{\boldsymbol{x}}}\right\|_2 \leq \mathcal{O}(B_{act} B_y^2 \epsilon),$$

*where $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ have been defined using $\boldsymbol{x}, \partial_{\boldsymbol{y}}$, and $\epsilon$ in Definitions F.2 and F.3.*

For ReLU activation, which is not second-order differentiable at 0, we instead bound the difference between $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ by defining some form of alignment between input and gradient pair $\boldsymbol{x}, \partial_{\boldsymbol{y}}$.

**Definition F.5** ($(\epsilon, \rho)$-alignment)**.** Input and gradient $\boldsymbol{x}, \partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{\text{aux}}}$ are said to be $(\epsilon, \rho)$-aligned, if there exist a set $C \subseteq [D_{\text{aux}}]$, with $|C| \geq (1-\rho) D_{\text{aux}}$, such that for each $i$ in $C$, $|x_i| > \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$.

$\epsilon$ controls the fraction of coordinates where $|x_i| \leq \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$. As $\epsilon \to 0$, $\rho \to 0$ as well for bounded gradients.

**Example F.6.** *For any $B_{min}, B_{max} > 0$, all inputs $\boldsymbol{x}$ that satisfy $\min_i |x_i| > B_{min}$ , and gradients $\partial_{\boldsymbol{y}}$ that satisfy $\max_j \left|(\partial_{\boldsymbol{y}})_j\right| \leq B_{max}$, are $(B_{min}/B_{max}, 0)$-aligned.*

**Theorem F.7.** *For any $\epsilon, \rho > 0$ and $B_y > 0$, for any input $\boldsymbol{x} \in \mathbb{R}^{D_{aux}}$ and gradient $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{aux}}$, with $\|\partial_{\boldsymbol{y}}\|_\infty \leq B_y$, that are $(\epsilon, \rho)$-aligned by Definition F.5,*

$$\left\|\partial_{\boldsymbol{x}} - \widehat{\partial_{\boldsymbol{x}}}\right\|_2 \leq \mathcal{O}(B_y \sqrt{\rho D_{aux}}).$$

*where $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ have been defined using $\boldsymbol{x}, \partial_{\boldsymbol{y}}$, $\epsilon$ and $\sigma_{act} = \mathrm{ReLU}$ in Definitions F.2 and F.3.*

**TINT Activation backpropagation module** The input embeddings contain $\partial_{\boldsymbol{y}_t}$ in the first $D_{\text{aux}}$ embeddings. With the requirement of the activation layer input for gradient, we copy $\boldsymbol{x}_t$ from the Forward module at each position $t$. We set $\epsilon$ as a hyper-parameter and return $\widehat{\partial_{\boldsymbol{x}_t}}$ as the output of this module.

$\widehat{\partial_{\boldsymbol{x}_t}}$ will be computed using a single-layer MLP with activation $\sigma_{\text{act}}$ as follows. The first linear layer of the MLP will be used to compute $\boldsymbol{x}_t + \epsilon \partial_{\boldsymbol{y}_t}$ and $\boldsymbol{x}_t$. After the activation $\sigma_{\text{act}}$, the embedding $\boldsymbol{e}_t$ contains $\sigma_{\text{act}}(\boldsymbol{x}_t + \epsilon \partial_{\boldsymbol{y}_t})$ and $\sigma_{\text{act}}(\boldsymbol{x}_t)$. The final linear layer of the MLP will be used to compute $\frac{1}{\epsilon}\left(\sigma_{\text{act}}(\boldsymbol{x}_t + \epsilon \partial_{\boldsymbol{y}_t}) - \sigma_{\text{act}}(\boldsymbol{x}_t)\right)$.

### F.1. Proofs of theorems

We restate the theorems, before presenting their proofs for easy referencing.

**Theorem F.4.** *For any $\epsilon > 0$, $B_y, B_{act} > 0$, consider a second-order differentiable activation function $\sigma_{act} : \mathbb{R} \to \mathbb{R}$, with $\partial^2 \sigma_{act}(x)/\partial(x^2)$ bounded by $B_{act}$ for each $x \in \mathbb{R}$. Then, for any input $\boldsymbol{x} \in \mathbb{R}^{D_{aux}}$ and gradient $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{aux}}$ with $\|\partial_{\boldsymbol{y}}\|_2 \leq B_y$, the following holds true:*

$$\left\|\partial_{\boldsymbol{x}} - \widehat{\partial_{\boldsymbol{x}}}\right\|_2 \leq \mathcal{O}(B_{act} B_y^2 \epsilon),$$

*where $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ have been defined using $\boldsymbol{x}, \partial_{\boldsymbol{y}}$, and $\epsilon$ in Definitions F.2 and F.3.*

*Proof.* The proof follows along the lines of Theorem E.4. Recall that given an input $\boldsymbol{x}$, the activation layer outputs $\boldsymbol{y} = \sigma_{\text{act}}(\boldsymbol{x})$, where the function $\sigma_{\text{act}}$ is applied coordinate-wise on $\boldsymbol{x}$. Given input $\boldsymbol{x}$ and the output gradient $\partial_{\boldsymbol{y}}$, the gradient w.r.t. the input is given by $\partial_{\boldsymbol{x}} = \sigma'_{\text{act}}(\boldsymbol{x}) \odot \partial_{\boldsymbol{y}}$, where the $\sigma'_{\text{act}}$ function is also applied coordinate wise to $\boldsymbol{x}$. We defined $\widehat{\partial_{\boldsymbol{x}}}$ as an $\epsilon$-approximate gradient, given by $\frac{1}{\epsilon}(\sigma_{\text{act}}(\boldsymbol{x} + \epsilon \partial_{\boldsymbol{y}}) - \sigma_{\text{act}}(\boldsymbol{x}))$. Since both $\sigma_{\text{act}}$ and $\sigma'_{\text{act}}$ are applied coordinate-wise, we can look at the coordinate-wise difference between $\partial_{\boldsymbol{x}}$ and $\widehat{\partial_{\boldsymbol{x}}}$.

Consider an arbitrary coordinate $i \leq D_{\text{aux}}$. Under the assumption that $\sigma_{\text{act}}$ is second-order differentiable, we have

$$\begin{aligned}
(\widehat{\partial_{\boldsymbol{x}}})_i &= \frac{1}{\epsilon}\left(\sigma_{\text{act}}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) - \sigma_{\text{act}}(x_i)\right) \\
&= \sigma'_{\text{act}}(x_i)(\partial_{\boldsymbol{y}})_i + \frac{1}{\epsilon}\int_{\theta=0}^{\epsilon} \frac{\partial^2 \sigma_{\text{act}}(x_\theta)}{\partial x_\theta^2} (\partial_{\boldsymbol{y}})_i^2 \theta d\theta \\
&= \sigma'_{\text{act}}(x_i)(\partial_{\boldsymbol{y}})_i + \mathcal{O}(\epsilon B_{act} (\partial_{\boldsymbol{y}})_i^2),
\end{aligned}$$

where $x_\theta$ represents $x_i + \theta(\partial_{\boldsymbol{y}})_i$ in the second step. In the final step, we utilize the upper bound assumption on $\frac{\partial^2 \sigma_{\text{act}}(x)}{\partial x^2}$.

Thus, $(\partial_{\boldsymbol{x}})_i - (\widehat{\partial_{\boldsymbol{x}}})_i = \mathcal{O}(\epsilon B_{act}(\partial_{\boldsymbol{y}})_i^2)$, and so

$$\left\|\partial_{\boldsymbol{x}} - \widehat{\partial_{\boldsymbol{x}}}\right\|_2 = \mathcal{O}(\epsilon B_{act} \sum_{i=1}^{D_{\text{aux}}} (\partial_{\boldsymbol{y}})_i^2) = \mathcal{O}(\epsilon B_{act} \left\|\partial_{\boldsymbol{y}}\right\|_2^2) \leq \mathcal{O}(\epsilon B_{act} B_y^2).$$

□

**Example F.6.** *For any $B_{min}, B_{max} > 0$, all inputs $\boldsymbol{x}$ that satisfy $\min_i |x_i| > B_{min}$ , and gradients $\partial_{\boldsymbol{y}}$ that satisfy $\max_j |(\partial_{\boldsymbol{y}})_j| \leq B_{max}$, are $(B_{min}/B_{max}, 0)$-aligned.*

*Proof.* Recall the definition of $(\epsilon, \rho)$-alignment from Definition F.5. Input and gradient $\boldsymbol{x}, \partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{\text{aux}}}$ are said to be $(\epsilon, \rho)$-aligned, if there exist a set $C \subseteq [D_{\text{aux}}]$, with $|C| \geq (1-\rho)D_{\text{aux}}$, such that for each $i$ in $C$, $|x_i| > \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$.

Consider an arbitrary coordinate $i \leq D_{\text{aux}}$. We have $|x_i| > \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$ for any $\epsilon < |x_i| / \left|(\partial_{\boldsymbol{y}})_i\right|$. Under the assumption that $|x_i| > B_{min}$, and $|(\partial_{\boldsymbol{y}})_i| \leq B_{max}$, a bound of $B_{min}/B_{max}$ suffices. □

**Theorem F.7.** *For any $\epsilon, \rho > 0$ and $B_y > 0$, for any input $\boldsymbol{x} \in \mathbb{R}^{D_{aux}}$ and gradient $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{aux}}$, with $\left\|\partial_{\boldsymbol{y}}\right\|_\infty \leq B_y$, that are $(\epsilon, \rho)$-aligned by Definition F.5,*

$$\left\|\partial_{\boldsymbol{x}} - \widehat{\partial_{\boldsymbol{x}}}\right\|_2 \leq \mathcal{O}(B_y \sqrt{\rho D_{aux}}).$$

*where $\partial_{\boldsymbol{x}}, \widehat{\partial_{\boldsymbol{x}}}$ have been defined using $\boldsymbol{x}, \partial_{\boldsymbol{y}}$, $\epsilon$ and $\sigma_{act} = \text{ReLU}$ in Definitions F.2 and F.3.*

*Proof.* Recall that given an input $\boldsymbol{x}$, the activation layer outputs $\boldsymbol{y} = \sigma_{\text{act}}(\boldsymbol{x})$, where the function $\sigma_{\text{act}}$ is applied coordinate-wise on $\boldsymbol{x}$. Given input $\boldsymbol{x}$ and the output gradient $\partial_{\boldsymbol{y}}$, the gradient w.r.t. the input is given by $\partial_{\boldsymbol{x}} = \sigma'_{\text{act}}(\boldsymbol{x}) \odot \partial_{\boldsymbol{y}}$, where the $\sigma'_{\text{act}}$ function is also applied coordinate wise to $\boldsymbol{x}$. We defined $\widehat{\partial_{\boldsymbol{x}}}$ as an $\epsilon$-approximate gradient, given by $\frac{1}{\epsilon}(\sigma_{\text{act}}(\boldsymbol{x} + \epsilon\partial_{\boldsymbol{y}}) - \sigma_{\text{act}}(\boldsymbol{x}))$. Since both $\sigma_{\text{act}}$ and $\sigma'_{\text{act}}$ are applied coordinate-wise, we can look at the coordinate-wise difference between $\partial_{\boldsymbol{x}}$ and $\widehat{\partial_{\boldsymbol{x}}}$. For ReLU activation, $\sigma'_{\text{act}}(x) = \text{sign}(x)$ for all $x \in \mathbb{R} \setminus \{0\}$, with $\sigma'_{\text{act}}(0) = 1$ to avoid ambiguity.

Going by the definition of $(\epsilon, \rho)$-alignment of the input and gradient from Definition F.5, we have a set $C$ with $|C| \geq (1-\rho)D_{\text{aux}}$ such that for each $i \in D_{\text{aux}}$, $|x_i| > \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$. For all coordinates $i \in C$, we can then observe that $\text{sign}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) = \text{sign}(x_i)$, implying

$$\sigma_{\text{act}}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) - \sigma_{\text{act}}(x_i) = \epsilon(\partial_{\boldsymbol{y}})_i \sigma'_{\text{act}}(x_i) = \epsilon(\partial_{\boldsymbol{x}})_i$$

For coordinates $i \notin C$, we have three possible cases:

- $\text{sign}(x_i) = \text{sign}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i)$: In this case, we can again show $\sigma_{\text{act}}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) - \sigma_{\text{act}}(x_i) = \epsilon(\partial_{\boldsymbol{y}})_i \sigma'_{\text{act}}(x_i) = \epsilon(\partial_{\boldsymbol{x}})_i$.
- $\text{sign}(x_i) = 0$, $\text{sign}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) = 1$: In this case, we have $\sigma'_{\text{act}}(x_i) = 0$, and so $(\partial_{\boldsymbol{x}})_i = 0$. Additionally, $\text{sign}((\partial_{\boldsymbol{y}})_i) = 1$, and so

  $$\left|\sigma_{\text{act}}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) - \sigma_{\text{act}}(x_i) - \epsilon(\partial_{\boldsymbol{x}})_i\right| = \left|x_i + \epsilon(\partial_{\boldsymbol{y}})_i\right| \leq \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|,$$

  where in the final step, we use the fact that $x_i < 0$ and $|x_i| < \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$.
- $\text{sign}(x_i) = 1$, $\text{sign}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) = 0$: In this case, we have $\sigma'_{\text{act}}(x_i) = 1$, and so $(\partial_{\boldsymbol{x}})_i = (\partial_{\boldsymbol{y}})_i$. Additionally, $\text{sign}((\partial_{\boldsymbol{y}})_i) = 0$, and so

  $$\left|\sigma_{\text{act}}(x_i + \epsilon(\partial_{\boldsymbol{y}})_i) - \sigma_{\text{act}}(x_i) - \epsilon(\partial_{\boldsymbol{x}})_i\right| = \left|-x_i - \epsilon(\partial_{\boldsymbol{y}})_i\right| \leq \left|\epsilon(\partial_{\boldsymbol{y}})_i\right|,$$

  where in the final step, we use the fact that $x_i \geq 0$ and $|x_i| < \epsilon \left|(\partial_{\boldsymbol{y}})_i\right|$.

Thus, from the above discussion, we have

$$\begin{aligned}\left\|\partial_{\boldsymbol{x}}-\widehat{\partial_{\boldsymbol{x}}}\right\|_2 &= \frac{1}{\epsilon}\left(\sum_{i=1}^{D_{\text{aux}}}(\sigma_{\text{act}}(x_i+\epsilon(\partial_{\boldsymbol{y}})_i)-\sigma_{\text{act}}(x_i)-\epsilon(\partial_{\boldsymbol{x}})_i)^2\right)^{1/2}\\ &= \frac{1}{\epsilon}\left(\sum_{i\notin C}(\sigma_{\text{act}}(x_i+\epsilon(\partial_{\boldsymbol{y}})_i)-\sigma_{\text{act}}(x_i)-\epsilon(\partial_{\boldsymbol{x}})_i)^2\right)^{1/2}\\ &\leq \left(\sum_{i\notin C}(\partial_{\boldsymbol{y}})_i^2\right)^{1/2} \leq \sqrt{\rho D_{\text{aux}}}\sqrt{\max_{i\notin C}(\partial_{\boldsymbol{y}})_i^2} \leq \sqrt{\rho D_{\text{aux}}}B_y.\end{aligned}$$

The final step includes a simple Cauchy Schwartz inequality and the desired bound comes from the assumed bound on $\|\partial_{\boldsymbol{y}}\|_2$. □

## G. Language model head

Additionally, we provide a description of the gradient computation for the loss function that involves the language model head. This computation entails performing a $\mathrm{softmax}$ operation over the entire vocabulary. If $\mathcal{V}$ denotes the vocabulary set of the auxiliary model, and $\boldsymbol{E}\in\mathbb{R}^{|\mathcal{V}|\times D_{\text{aux}}}$ denotes the embedding matrix of the auxiliary model, we directly utilize the embedding matrix for the auto-regressive loss in the TINT. Additionally, we do not update the embedding matrix of the auxiliary model; instead, we solely backpropagate the gradients through the language model head. Recent work in (Kumar et al., 2022) has shown that keeping the embedding matrix fixed while updating the model can stabilize SGD. We demonstrate that the backpropagated gradients can be expressed as the combination of the language model head and a self-attention layer.

**Definition G.1** (KL-loss gradient through auxiliary's language model head)**.** Given an embedding matrix $\boldsymbol{E}\in\mathbb{R}^{|V|\times D_{\text{aux}}}$, the language model head takes in input $\boldsymbol{x}\in\mathbb{R}^{D_{\text{aux}}}$ and a target distribution $\boldsymbol{q}\in\mathbb{R}^{|V|}$ and returns gradient $\partial_{\boldsymbol{x}}\in\mathbb{R}^{D_{\text{aux}}}$, with $\partial_{\boldsymbol{x}}=\boldsymbol{E}^\top\left(\mathrm{softmax}(\boldsymbol{E}\boldsymbol{x})-\boldsymbol{q}\right)$.

In the autoregressive loss on a sequence of tokens, the target output distribution at any position is the next occurring token. If $\{\boldsymbol{x}_t^{un}\}_{t=1}^{T_{\text{aux}}}$ denote the uncontextualized embeddings of a sequence of tokens after encoding them via the embedding matrix, and $\{\boldsymbol{x}_t\}_{t=1}^{T_{\text{aux}}}$ denote their contextualized embeddings after passing through the auxiliary model, then the gradient $\partial_{\boldsymbol{x}_t}$ at any position $t$ can be simplified as $\boldsymbol{E}^\top\mathrm{softmax}(\boldsymbol{E}\boldsymbol{x}_t)-\boldsymbol{x}_{t+1}^{un}$. We illustrate the involved TINT module w.r.t. an arbitrary position $t$.

**TINT autoregressive loss gradient module** The current embedding $\boldsymbol{e}_t$ contains the contextualized embedding $\boldsymbol{x}_t$ in its first $D_{\text{aux}}$ coordinates. Furthermore, $\boldsymbol{e}_t$ includes the uncontextualized embedding $\boldsymbol{x}_t^{un}$, copied from the input layer using residual connections. The prefix tokens $\boldsymbol{v}_j$ are assigned a value of 0 and do not participate in the subsequent computations.

The loss computation can be decomposed into two sub-operations: (a) computing $\boldsymbol{y}_t:=\boldsymbol{E}^\top\mathrm{softmax}(\boldsymbol{E}\boldsymbol{x}_t)$, and (b) calculating $\partial_{\boldsymbol{x}_t}=\boldsymbol{y}_t-\boldsymbol{x}_{t+1}^{un}$.

For the first sub-operation, we use a feed-forward layer with $\mathrm{softmax}$ activation, with hidden and output weights $\boldsymbol{E}$ and $\boldsymbol{E}^\top$ respectively, that takes in the first $D_{\text{aux}}$ of $\boldsymbol{e}_t$ and returns $\boldsymbol{y}_t$ in the first $D_{\text{aux}}$ coordinates. We retain $\boldsymbol{x}_t^{un}$ using a residual connection.

The final sub-operation can be interpreted as a TINT self-attention layer. With $\boldsymbol{e}_t$ containing both $\boldsymbol{y}_t$ and $\boldsymbol{x}_t^{un}$, we use a linear self-attention layer (Definition B.2) with two attention heads. The first attention head assigns an attention score of 1 to pairs $\{(t,t+1)\}_{t\leq T_{\text{aux}}-1}$, while assigning an attention score of 0 to the remaining pairs. At any position $t$, $-\boldsymbol{x}_t^{un}$ is considered the value vector. The second attention head assigns an attention score of 1 to pairs $\{(t,t)\}_{t\leq T_{\text{aux}}}$, while assigning an attention score of 0 to the remaining pairs. At any position $t$, $\boldsymbol{y}_t$ is considered the value vector. The outputs of both attention heads are subsequently combined using a linear layer.

*Remark* G.2. We conducted experiments using mean-squared loss and Quad loss (Saunshi et al., 2020), which do not necessitate softmax computations for gradient computation. As an example, in the case of mean-squared loss, if our objective is to minimize $\frac{1}{2}\sum_{t=1}^{T}\left\|\boldsymbol{x}_t-\boldsymbol{x}_{t+1}^{un}\right\|^2$, the gradient can be computed as $\partial_{\boldsymbol{x}_t}=\boldsymbol{x}_t-\boldsymbol{x}_{t+1}^{un}$. Similarly, in the case of Quad loss, the gradient is $\partial_{\boldsymbol{x}_t}=\frac{1}{|V|}\sum_i\boldsymbol{e}_i-\boldsymbol{x}_{t+1}^{un}$. However, in all of our language model experiments (Section 5), both gradients

resulted in minimal improvement in perplexity compared to the auxiliary model. Therefore, we continue utilizing the standard KL loss for optimization.

*Remark* G.3. For ease of implementation in the codebase, we utilize a dedicated loss module that takes in $\boldsymbol{y}_t, \boldsymbol{x}_{t+1}^{un}$ as input and directly computes $\partial_{\boldsymbol{x}_t} = \boldsymbol{y}_t - \boldsymbol{x}_{t+1}^{un}$.

## H. Parameter sharing

**Feed-forward layer of auxiliary model:** In a standard auxiliary transformer, like GPT-2, the feed-forward layer is a token-wise operation that takes in an input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ and returns $\boldsymbol{y} = \boldsymbol{A}\sigma(\boldsymbol{W}\boldsymbol{x})$, with $\boldsymbol{A} \in \mathbb{R}^{D_{\text{aux}} \times 4D_{\text{aux}}}$ and $\boldsymbol{W} \in \mathbb{R}^{4D_{\text{aux}} \times D_{\text{aux}}}$. A naive construction of the TINTto simulate its forward operation will have 2 Linear Forward modules (Section 3), separated by an activation. However, this requires $4\times$ more prefix embeddings to represent the parameters, compared to other linear operations in the auxiliary transformer that use $\mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ weight parameters.

To avoid this, we can instead break down the computation into 4 sub-feed-forward layers, each with its own parameters $\{\{\boldsymbol{W}^i, \boldsymbol{A}^i\}\}_{1 \le i \le 4}$. Here $\{\boldsymbol{W}^i\}_{1 \le i \le 4}$ represent 4-shards of the rows of $\boldsymbol{W}$, and $\{\boldsymbol{A}^i\}_{1 \le i \le 4}$ represent 4-shards of the columns of $\boldsymbol{A}$.

The forward, backward, and descent operations on these 4 sub-feed-forward layers can be effectively parallelized. For example, the forward operation of each layer can be simulated by a single TINTmodule, consisting of two Linear Forward modules and activation, changing only the prefix embeddings to correspond to $\{\{\boldsymbol{W}^i, \boldsymbol{A}^i\}\}_{1 \le i \le 4}$.

## I. Additional modules

We describe the forward, backward, and decent update operations of additional modules, used in different model families, like LLaMA (Touvron et al., 2023) and BLOOM (Scao et al., 2022). We discuss the simulation of these modules, using similar TINT modules.

### I.1. Root mean square normalization (RMSnorm)

The operation of RMSnorm (Zhang and Sennrich, 2019) is very similar to layer normalization.

**Definition I.1** (RMSnorm)**.** For an arbitrary dimension $d$, define a normalization function $f : \mathbb{R}^d \to \mathbb{R}^d$ that performs $f(\boldsymbol{x}) = \boldsymbol{x}/RMS(\boldsymbol{x})$, where $RMS(\boldsymbol{x}) = (\sum_{i=1}^d x_i^2)^{1/2}$. Then, RMSnorm with parameters $\gamma, \boldsymbol{b} \in \mathbb{R}^{D_{\text{aux}}}$ takes as input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ and outputs $\boldsymbol{y} \in \mathbb{R}^{D_{\text{aux}}}$, which is computed as $\boldsymbol{z} = f(\boldsymbol{x}), \boldsymbol{y} = \gamma \odot \boldsymbol{z} + \boldsymbol{b}$.

The extreme similarity between RMSnorm and layer normalization (Definition E.1) helps us create similar TINT modules as described in Appendix E, where instead of Group normalization layers, we use Group RMSnorm layers described below.

**Definition I.2** (TINT $D_{\text{aux}}$-Group RMSnorm)**.** For an arbitrary dimension $d$, define a normalization function $f : \mathbb{R}^d \to \mathbb{R}^d$ that performs $f(\boldsymbol{x}) = \boldsymbol{x}/RMS(\boldsymbol{x})$, where $RMS(\boldsymbol{x}) = (\sum_{i=1}^d x_i^2)^{1/2}$. Then, $D_{\text{aux}}$-Group RMSnorm with parameters $\gamma^{\text{TINT}}, \boldsymbol{b}^{\text{TINT}} \in \mathbb{R}^{D_{\text{aux}}}$ takes as input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{sim}}}$ and outputs $\boldsymbol{y} = \text{VECTORIZE}(\{\boldsymbol{y}^h \in \mathbb{R}^{D_{\text{aux}}}\}_{h \le \lfloor D_{\text{sim}}/D_{\text{aux}} \rfloor})$, with

$$\boldsymbol{y}^h = \gamma^{\text{TINT}} \odot f(\boldsymbol{x}^h) + \boldsymbol{b}^{\text{TINT}},$$

where $\boldsymbol{x}^h = \text{SPLIT}_{\lfloor D_{\text{sim}}/D_{\text{aux}} \rfloor}(\boldsymbol{x})_h$.

### I.2. Attention variants

In order to incorporate additional attention variants, e.g. Attention with Linear Biases (ALiBi) (Press et al., 2021), and rotary position embeddings (Su et al., 2021), we can change the definition of softmax attention layer in Definition B.2 likewise.

We showcase the changes for ALiBi.

**Definition I.3** (Auxiliary ALiBi self-attention with $H_{\text{aux}}$ heads)**.** For query, key, and value weights $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$, bias $\boldsymbol{b}_Q, \boldsymbol{b}_K, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$ and $\boldsymbol{m} \in \mathbb{R}^{H_{\text{aux}}}$, ALiBi self-attention layer with $H_{\text{aux}}$ attention heads and a function $f_{\text{attn}} : \mathbb{R}^{T_{\text{aux}}} \to \mathbb{R}^{T_{\text{aux}}}$ takes a sequence $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t \le T_{\text{aux}}}$ as input and outputs $\{\boldsymbol{y}_t\}_{t \le T_{\text{aux}}}$, with

$$\boldsymbol{y}_t = \text{VECTORIZE}(\{\sum_{j \le T_{\text{aux}}} a_{t,j}^h \boldsymbol{v}_j^h\}_{h \le H_{\text{aux}}}). \tag{23}$$

$a^h_{t,j}$ is defined as the attention score of head $h$ between tokens at positions $t$ and $j$, and is given by

$$a^h_{t,j} = \text{softmax}(\boldsymbol{K}^h \boldsymbol{q}^h_t + m_h \boldsymbol{r}_t)_j. \tag{24}$$

Here $\boldsymbol{r}_t \in \mathbb{R}^{T_{\text{aux}}}$ denotes a relative position vector at each position $t$ that contains $(j-t)$ at each coordinate $j \le T_{\text{aux}}$. Here, $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ denote the query, key, and value vectors at each position $t$, computed as $\boldsymbol{W}_Q \boldsymbol{x}_t + \boldsymbol{b}_Q$, $\boldsymbol{W}_K \boldsymbol{x}_t + \boldsymbol{b}_K$, and $\boldsymbol{W}_V \boldsymbol{x}_t + \boldsymbol{b}_V$ respectively. In addition, $\boldsymbol{q}^h_t, \boldsymbol{k}^h_t, \boldsymbol{v}^h_t$ denote $\text{SPLIT}_{H_{\text{aux}}}(\boldsymbol{q}_t)_h$, $\text{SPLIT}_{H_{\text{aux}}}(\boldsymbol{k}_t)_h$, and $\text{SPLIT}_{H_{\text{aux}}}(\boldsymbol{v}_t)_h$ respectively for all $t \le T_{\text{aux}}$, and $h \le H_{\text{aux}}$. $\boldsymbol{K}^h \in \mathbb{R}^{T_{\text{aux}} \times D_{\text{aux}}}$ is defined with its rows as $\{\boldsymbol{k}^h_t\}_{t \le T_{\text{aux}}}$ for all $h \le H_{\text{aux}}$.

To include operations involving ALiBi, we modify the self-attention module of TINT to change the definition of the attention scores like Equation (24).

**Definition I.4** (Modified TINT self-attention for ALiBi with $H_{\text{sim}}$ heads)**.** For parameters $\{\boldsymbol{W}_Q^{\text{TINT}}, \boldsymbol{W}_K^{\text{TINT}}, \boldsymbol{W}_V^{\text{TINT}} \in \mathbb{R}^{D_{\text{sim}} \times D_{\text{sim}}}\}$, $\{\boldsymbol{b}_Q^{\text{TINT}}, \boldsymbol{b}_K^{\text{TINT}}, \boldsymbol{b}_V^{\text{TINT}} \in \mathbb{R}^{D_{\text{sim}}}\}$, $\{\boldsymbol{W}_Q^p, \boldsymbol{W}_K^p, \boldsymbol{W}_V^p \in \mathbb{R}^{T_{\text{sim}} \times D_{\text{sim}}/H_{\text{sim}}}\}$, $\{\lambda^Q, \lambda^K, \lambda^V \in \mathbb{R}^{H_{\text{sim}}}\}$ and $\boldsymbol{m}^{\text{TINT}} \in \mathbb{R}^{T_{\text{sim}}}$, TINT self-attention with $H_{\text{sim}}$ attention heads and a function $f_{\text{attn}} : \mathbb{R}^{T_{\text{sim}}} \to \mathbb{R}^{T_{\text{sim}}}$ takes a sequence $\{\widehat{\boldsymbol{e}}_t \in \mathbb{R}^{D_{\text{sim}}}\}_{t \le T_{\text{sim}}}$ as input and outputs $\{\widetilde{\boldsymbol{e}}_t \in \mathbb{R}^{D_{\text{sim}}}\}_{t \le T_{\text{sim}}}$, with

$$\begin{aligned}
&\widetilde{\boldsymbol{e}}_t = \text{VECTORIZE}(\{\sum_{j \le T_{\text{sim}}} a^h_{t,j} \widetilde{\boldsymbol{v}}^h_j)_h\}_{h \le H_{\text{sim}}}), \text{ with } a^h_{t,j} = f_{\text{attn}}(\widetilde{\boldsymbol{K}}^h \widetilde{\boldsymbol{q}}^h_t + m^{\text{TINT}}_h \boldsymbol{r}_t)_j \\
&\widetilde{\boldsymbol{q}}^h_t = \text{SPLIT}_H(\boldsymbol{q}_t)_h + \lambda^Q_h \boldsymbol{W}^p_Q \boldsymbol{p}^{\text{TINT}}_t; \quad \widetilde{\boldsymbol{k}}^h_t = \text{SPLIT}_H(\boldsymbol{k}_t)_h + \lambda^K_h \boldsymbol{W}^p_K \boldsymbol{p}^{\text{TINT}}_t +; \\
&\widetilde{\boldsymbol{v}}^h_t = \text{SPLIT}_H(\boldsymbol{v}_t)_h + \lambda^V_h \boldsymbol{W}^p_v \boldsymbol{p}^{\text{TINT}}_t.
\end{aligned}$$

Here $\boldsymbol{r}_t \in \mathbb{R}^{T_{\text{sim}}}$ denotes a relative position vector at each position $t$ that contains $(j-t)$ at each coordinate $j \le T_{\text{sim}}$. Here, $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t$ denote the query, key, and value vectors at each position $t$, computed as $\boldsymbol{W}_Q^{\text{TINT}} \widehat{\boldsymbol{e}}_t + \boldsymbol{b}_Q^{\text{TINT}}$, $\boldsymbol{W}_K^{\text{TINT}} \widehat{\boldsymbol{e}}_t + \boldsymbol{b}_K^{\text{TINT}}$, and $\boldsymbol{W}_V^{\text{TINT}} \widehat{\boldsymbol{e}}_t + \boldsymbol{b}_V^{\text{TINT}}$ respectively. $\widetilde{\boldsymbol{K}}^h \in \mathbb{R}^{T_{\text{sim}} \times D_{\text{sim}}/H_{\text{sim}}}$ is defined with its rows as $\{\widetilde{\boldsymbol{k}}^h_t\}_{t \le T_{\text{sim}}}$ for all $h \le H_{\text{sim}}$.

After referring to Appendix D, we make the following modifications to the Forward, Backward, and Descent modules. In the Forward module, we incorporate the modified self-attention module to compute the attention scores using ALiBi attention. In the Backward module, since we do not propagate gradients through the attention scores of the auxiliary model, the backpropagation formulation remains unchanged from Definition D.3 when we have access to the attention scores. Similarly, in the Descent module, we update the value matrix while keeping the query and key parameters fixed. The formulation of the gradient update remains unchanged from Definition D.6 when we have access to the attention scores. Consequently, we simply modify all the self-attention modules in the simulator to include ALiBi attention, as defined by Definition I.4.

### I.3. Gated linear units (GLUs)

We describe the operations of GLUs (Shazeer, 2020) using similar GLU units available to the TINT.

**Definition I.5.** For parameters $\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{W}^o \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$, and biases $\boldsymbol{b}_W, \boldsymbol{b}_V, \boldsymbol{b}_{W^o} \in \mathbb{R}^{D_{\text{aux}}}$, a GLU layer with activation $\sigma_{\text{act}} : \mathbb{R} \to \mathbb{R}$, takes input $\boldsymbol{x} \in \mathbb{R}^{D_{\text{aux}}}$ and outputs $\widehat{\boldsymbol{y}} \in \mathbb{R}^{D_{\text{aux}}}$, with

$$\boldsymbol{y} = (\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}_V); \quad \widehat{\boldsymbol{y}} = \boldsymbol{W}^o \boldsymbol{y} + \boldsymbol{b}_{W^o}.$$

Typical GLUs have $8/3 \times D_{\text{aux}}$ as a hidden dimension (i.e. the dimension of $\boldsymbol{y}$). We can use similar parameter-sharing techniques discussed for feed-forward layers (Appendix H) with the TINT modules presented here. Furthermore, since $\widehat{y}$ can be expressed as a combination of the gated operation and a linear operation, we focus on the computation of $\boldsymbol{y}$ here.

For the discussion below, we consider a GLU (without the output linear layer) in the auxiliary model, with parameters $\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{b}_W, \boldsymbol{b}_V$, that takes in input sequence $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T$ and outputs $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_T$, with $\boldsymbol{y}_t = (\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V)$ for each $t \le T_{\text{sim}}$. Since this involves a token-wise operation, we will present our constructed modules with a general token position $t$ and the prefix tokens $\{\boldsymbol{v}_j\}$.

**TINT GLU Forward module** The embedding $\boldsymbol{e}_t$ contains $\boldsymbol{x}_t$ in its first $D_{\text{aux}}$ coordinates. The output $\boldsymbol{y}_t$ can be computed using three sub-operations: (a) linear operation for $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W$, (b) linear operation for $\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V$, and (c) gate operation to get $(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V)$.

We use three TINT modules, representing each sub-operation.

(a) $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W$ is a linear operation, hence we can use a TINT Linear Forward module (Appendix C) with the current embedding $\boldsymbol{e}_t$ and $\{\boldsymbol{v}_j\}$ containing $\boldsymbol{W}, \boldsymbol{b}_W$ to get embedding $\widetilde{\boldsymbol{e}}_t$ containing $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W$ in its first $D_{\text{aux}}$ coordinates.

(b) $\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V$ is a linear operation, hence we can similarly use a TINT Linear Forward module (Appendix C) with the embedding $\boldsymbol{e}_t$ and $\{\boldsymbol{v}_j\}$ containing $\boldsymbol{W}_V, \boldsymbol{b}_V$ to get embedding $\widehat{\boldsymbol{e}}_t$ containing $\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V$ in its first $D_{\text{aux}}$ coordinates. $\widehat{\boldsymbol{e}}_t$ and $\widetilde{\boldsymbol{e}}_t$ are now combined to get an embedding $\boldsymbol{e}_t$ that contains $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W, \boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V$ in its first $2D_{\text{aux}}$ coordinates.

(c) Finally, we can use a TINT GLU layer that can carry out the elementwise multiplication of $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W, \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V)$ to get $\boldsymbol{y}_t$ in the first $D_{\text{aux}}$ coordinates.

*Parameter Sharing:* Since (a) and (b) involve a Linear Forward module, we can additionally leverage parameter sharing to apply a single Linear Forward module for each of the two computations, changing only the prefix embeddings to correspond to $\boldsymbol{W}, \boldsymbol{b}_W$, or $\boldsymbol{W}_V, \boldsymbol{b}_V$.

**Auxiliary GLU backpropagation** For the GLU layer defined in Definition I.5, the backpropagation layer takes in the loss gradient w.r.t. output ($\partial_{\boldsymbol{y}}$) and computes the loss gradient w.r.t. input ($\partial_{\boldsymbol{x}}$).

**Definition I.6** (Auxiliary GLU backpropagation)**.** For the weights $\boldsymbol{W}, \boldsymbol{V} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ , the backpropagation layer takes $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\partial_{\boldsymbol{x}} \in \mathbb{R}^{D_{\text{aux}}}$, with $\partial_{\boldsymbol{x}} = \boldsymbol{W}^\top \widehat{\partial_{\boldsymbol{x}}} + \boldsymbol{V}^\top \widetilde{\partial_{\boldsymbol{x}}}$, where

$$\widehat{\partial_{\boldsymbol{x}}} = \partial_{\boldsymbol{y}} \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}_V); \qquad \widetilde{\partial_{\boldsymbol{x}}} = \sigma'_{\text{act}}(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}_V) \odot \partial_{\boldsymbol{y}} \odot (\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_W).$$

A direct computation of $\widetilde{\partial_{\boldsymbol{x}}}$ involves changing the activation function to $\sigma'_{\text{act}}$. Following a similar strategy for backpropagation through an activation layer (Appendix F), we instead use a first-order Taylor expansion to approximate $\widetilde{\partial_{\boldsymbol{x}}}$.

**Definition I.7** (Auxiliary GLU approximate backpropagation)**.** For a hyper-parameter $\epsilon > 0$, for the weights $\boldsymbol{W}, \boldsymbol{V} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ , the approximate backpropagation layer takes $\partial_{\boldsymbol{y}} \in \mathbb{R}^{D_{\text{aux}}}$ as input and outputs $\overline{\partial_{\boldsymbol{x}}} \in \mathbb{R}^{D_{\text{aux}}}$, with $\overline{\partial_{\boldsymbol{x}}} = \boldsymbol{W}^\top \widehat{\partial_{\boldsymbol{x}}} + \boldsymbol{V}^\top \widehat{\widetilde{\partial_{\boldsymbol{x}}}}$, where

$$\begin{aligned}
\widehat{\partial_{\boldsymbol{x}}} &= \partial_{\boldsymbol{y}} \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}_V) \\
\widehat{\widetilde{\partial_{\boldsymbol{x}}}} &= \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}_V + \epsilon\partial_{\boldsymbol{y}}) \odot \frac{1}{\epsilon}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_W) - \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x} + \boldsymbol{b}_V) \odot \frac{1}{\epsilon}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_W).
\end{aligned}$$

**TINT GLU backpropagation module** The current embedding contains $\partial_{\boldsymbol{y}_t}$ in its first $D_{\text{aux}}$ coordinates. Furthermore, since we need $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W$ and $\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V$ in the gradient computations, we copy them from the Forward module using residual connections. We discuss the computation of $\boldsymbol{W}^\top \widehat{\partial_{\boldsymbol{x}_t}}$ and $\boldsymbol{V}^\top \widehat{\widetilde{\partial_{\boldsymbol{x}_t}}}$ as separate sub-modules acting on the same embedding $\boldsymbol{e}_t$ in parallel.

1. The computation of $\boldsymbol{W}^\top \widehat{\boldsymbol{x}_t}$ involves two sub-operations: (a) gate operation to get $\widehat{\boldsymbol{x}_t} := \partial_{\boldsymbol{y}_t} \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V)$, and (b) linear backward operation to get $\boldsymbol{W}^\top \widehat{\boldsymbol{x}_t}$. Since for this operation, we require $\boldsymbol{W}$, we copy the contents of the prefix embeddings containing $\boldsymbol{W}, \boldsymbol{b}_W$ from the Forward module.
   (a) Since the current embedding $\boldsymbol{e}_t$ contains both $\partial_{\boldsymbol{y}_t}$ and $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W$, we can use a TINT GLU layer to get an embedding $\widehat{\boldsymbol{e}}_t^{(1)}$ that contains $\widehat{\partial_{\boldsymbol{x}_t}}$.
   (b) The final linear backward operation can be performed by using a TINT Linear backpropagation module (Appendix C) with the embeddings $\widehat{\boldsymbol{e}}_t^{(1)}$ and the prefix embeddings. The final embedding $\widehat{\boldsymbol{e}}_t$ contains $\boldsymbol{W}^\top \widehat{\boldsymbol{x}_t}$ in the first $D_{\text{aux}}$ coordinates.
2. The computation of $\boldsymbol{V}^\top \widehat{\widetilde{\boldsymbol{x}_t}}$ involves four sub-operations: (a) gate operation to get $\frac{1}{\epsilon}(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V + \epsilon\partial_{\boldsymbol{y}_t})$, (b) gate operation to get $\frac{1}{\epsilon}(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V)$, (c) a linear layer to compute $\widehat{\widetilde{\boldsymbol{x}_t}}$, (c) linear backward operation to get $\boldsymbol{V}^\top \widehat{\widetilde{\boldsymbol{x}_t}}$. Since for this operation, we require $\boldsymbol{V}$, we copy the contents of the prefix embeddings containing $\boldsymbol{V}, \boldsymbol{b}_V$ from the Forward module.
   (a) Since the current embedding $\boldsymbol{e}_t$ contains $\partial_{\boldsymbol{y}_t}$, $\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_W$ and $\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W$, we can use two TINT GLU layers to get an embedding $\widetilde{\boldsymbol{e}}_t^{(1)}$ that contains both $\frac{1}{\epsilon}(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V + \epsilon\partial_{\boldsymbol{y}_t})$ and $\frac{1}{\epsilon}(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b}_W) \odot \sigma_{\text{act}}(\boldsymbol{V}\boldsymbol{x}_t + \boldsymbol{b}_V)$.

(b) A linear later on $\widetilde{\boldsymbol{e}}_t^{(1)}$ can then return an embedding $\widetilde{\boldsymbol{e}}_t^{(2)}$ containing $\widehat{\widetilde{\boldsymbol{x}}_t}$ in the first $D_{\text{aux}}$ coordinates.

(c) The final operation can be performed by using a TINT Linear backpropagation module (Appendix C) with the embeddings $\widehat{\boldsymbol{e}}_t^2$ and the prefix embeddings containing $\boldsymbol{V}, \boldsymbol{b}_V$. The final embedding $\widetilde{\boldsymbol{e}}_t$ contains $\boldsymbol{V}^\top \widehat{\widetilde{\boldsymbol{x}}_t}$ in the first $D_{\text{aux}}$ coordinates.

After the two parallel computations, we can sum up $\widehat{\boldsymbol{e}}_t$ and $\widetilde{\boldsymbol{e}}_t$ to get an embedding $\boldsymbol{e}_t$ containing $\overline{\partial_{\boldsymbol{x}_t}}$ (Definition I.7) in the first $D_{\text{aux}}$ coordinates.

**Auxiliary GLU descent** Finally, the auxiliary's descent updates the weight and the bias parameters using a batch of inputs $\{\boldsymbol{x}_t\}_{t\le T}$ and the loss gradient w.r.t. the corresponding outputs $\{\partial_{\boldsymbol{y}_t}\}_{t\le T}$.

**Definition I.8** (Auxiliary GLU descent )**.** For weights $\boldsymbol{W}, \boldsymbol{V} \in \mathbb{R}^{D_{\text{aux}} \times D_{\text{aux}}}$ and bias $\boldsymbol{b}_W, \boldsymbol{b}_V \in \mathbb{R}^{D_{\text{aux}}}$, the linear descent layer takes in a batch of inputs $\{\boldsymbol{x}_t \in \mathbb{R}^{D_{\text{aux}}}\}_{t\le T_{\text{aux}}}$ and gradients $\{\partial_{\boldsymbol{y}_t} \in \mathbb{R}^{D_{\text{aux}}}\}_{t\le T_{\text{aux}}}$ and updates the parameters as follows:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \sum_{t\le T_{\text{aux}}} \widehat{\partial_{\boldsymbol{x}_t}} \boldsymbol{x}_t^\top; \qquad \boldsymbol{b}_W \leftarrow \boldsymbol{b}_W - \eta \sum_{t\le T_{\text{aux}}} \widehat{\partial_{\boldsymbol{x}_t}},$$
$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \eta \sum_{t\le T_{\text{aux}}} \widetilde{\partial_{\boldsymbol{x}_t}} \boldsymbol{x}_t^\top; \qquad \boldsymbol{b}_V \leftarrow \boldsymbol{b}_V - \eta \sum_{t\le T_{\text{aux}}} \widetilde{\partial_{\boldsymbol{x}_t}},$$

where $\widehat{\partial_{\boldsymbol{x}_t}}$ and $\widetilde{\partial_{\boldsymbol{x}_t}}$ have been computed as Definition I.6.

Due to similar concerns as gradient backpropagation, we instead use $\widehat{\widetilde{\partial_{\boldsymbol{x}_t}}}$ (Definition I.7) in place of $\widetilde{\partial_{\boldsymbol{x}_t}}$ for each $t \le T_{\text{aux}}$ to update $\boldsymbol{V}, \boldsymbol{b}_V$.

**TINT GLU descent module** We discuss the two descent operations separately.

1. Update of $\boldsymbol{W}, \boldsymbol{b}_W$: We start with the embeddings $\widehat{\boldsymbol{e}}_t^{(1)}$ from the backpropagation module, that contain $\widehat{\partial_{\boldsymbol{x}_t}}$ in the first $D_{\text{aux}}$ coordinates.

   For the update, we additionally require the input to the auxiliary GLU layer under consideration, and hence we copy $\boldsymbol{x}_t$ from the Forward module using residual connections. Furthermore, we copy the contents of the prefix embeddings that contain $\boldsymbol{W}, \boldsymbol{b}_W$ from the Forward module.

   With both $\widehat{\partial_{\boldsymbol{x}_t}}$ and $\boldsymbol{x}_t$ in the embeddings, the necessary operation turns out to be the descent update of a linear layer with parameters $\boldsymbol{W}, \boldsymbol{b}_W$. That implies, we can call a TINT Linear descent module (Appendix C) on the current embeddings and prefix embeddings to get the desired update.

2. We start with the embeddings $\widetilde{\boldsymbol{e}}_t^{(2)}$ from the backpropagation module, that contain $\widehat{\widetilde{\partial_{\boldsymbol{x}_t}}}$ in the first $D_{\text{aux}}$ coordinates.

   For the update, we additionally require the input to the auxiliary GLU layer under consideration, and hence we copy $\boldsymbol{x}_t$ from the forward module using residual connections. Furthermore, we copy the contents of the prefix embeddings that contain $\boldsymbol{V}, \boldsymbol{b}_V$ from the Forward module.

   With both $\widehat{\widetilde{\partial_{\boldsymbol{x}_t}}}$ and $\boldsymbol{x}_t$ in the embeddings, the necessary operation turns out to be the descent update of a linear layer with parameters $\boldsymbol{V}, \boldsymbol{b}_V$. That implies we can call a TINT Linear descent module on the current embeddings and prefix embeddings to get the desired update.

*Parameter sharing*: Since both the descent updates involve a Linear descent module, we can additionally leverage parameter sharing to apply a single TINT Linear descent module for each of the two computations, changing the input to correspond to $\{\widehat{\boldsymbol{e}}_t^{(1)}\}$ and prefix to correspond to $\boldsymbol{W}, \boldsymbol{b}_W$, or the input to correspond to $\{\widetilde{\boldsymbol{e}}_t^{(2)}\}$ and prefix to correspond to $\boldsymbol{V}, \boldsymbol{b}_V$ respectively.

## J. Construction of other variants of pre-trained models

Though we only conduct experiments on an OPT-125M model, our construction is generally applicable to diverse variants of pre-trained language models. Table 3 highlights many types of modules and the required size and computation for each. The size of a constructed model is influenced by various factors, including the number of layers, and embedding dimension in the auxiliary.

## K. Experiments

**Computing environment**: All the experiments are conducted on a single A100 80G GPU.

**Hyperparameters:** In the few-shot setting, we employ three different random seeds to select distinct sets of training examples. Grid search is performed for each seed to determine the optimal learning rate for both constructed models and dynamic evaluation. The learning rates considered for the learning rate hyperparameter in the descent update operations in TINT are $1e-3, 1e-4, 1e-5$. [9] Additionally, we explore various layer-step combinations to allocate a fixed budget for one full forward pass. Specifically, we update the top 3 layers for 4 steps, the top 6 layers for 3 steps, or 12 layers for 1 step. These specific combinations were chosen to demonstrate the flexibility of TinT in simulating fine-tuning for any number of layers and steps while staying within computational constraints. In all of these scenarios, TINT performs as well as fine-tuning the auxiliary model.

**Results of different settings.** Table 4 displays the results of few-shot learning with calibration across various settings, encompassing different loss types, input formats, and layer-step configurations. Our analysis reveals that employing a label-only loss, utilizing a single-example input format, and updating all layers of the internal model for a single step yield the most favorable average result. The performance of the multi-example format is disadvantaged when dealing with tasks of long sequences such as Amazon Polarity. In general, we observe that calibrated results tend to be more consistent and stable.

Table 4: Few-shot ($k = 32$) results with different loss types, input formats, and layer-step configurations with a fixed compute budget, with calibration.

| Loss Type | Format | Layer | Step | Subj | AGNews | SST2 | CR | MR | MPQA | Amazon | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | Single | 12 | 1 | $66.0_{(1.9)}$ | $64.7_{(0.2)}$ | $68.7_{(1.3)}$ | $69.0_{(0.7)}$ | $63.7_{(0.2)}$ | $82.8_{(0.5)}$ | $73.7_{(0.6)}$ | $69.8_{(0.1)}$ |
| | Single | 6 | 2 | $62.7_{(0.2)}$ | $66.3_{(0.2)}$ | $68.3_{(6.1)}$ | $67.2_{(0.2)}$ | $61.8_{(1.6)}$ | $81.0_{(3.6)}$ | $74.3_{(0.5)}$ | $68.8_{(1.4)}$ |
| | Single | 3 | 4 | $63.5_{(0.0)}$ | $67.2_{(0.8)}$ | $62.5_{(0.4)}$ | $68.7_{(1.4)}$ | $61.7_{(0.6)}$ | $76.8_{(3.3)}$ | $75.2_{(0.8)}$ | $67.9_{(0.8)}$ |
| | Multi. | 12 | 1 | $83.2_{(2.5)}$ | $43.7_{(6.6)}$ | $60.7_{(5.7)}$ | $70.3_{(6.1)}$ | $62.8_{(8.9)}$ | $84.2_{(1.6)}$ | $66.3_{(12.3)}$ | $67.3_{(0.9)}$ |
| | Multi. | 6 | 2 | $83.5_{(2.9)}$ | $43.2_{(8.4)}$ | $52.0_{(1.5)}$ | $70.5_{(6.0)}$ | $58.5_{(11.3)}$ | $82.0_{(0.4)}$ | $55.8_{(7.6)}$ | $63.6_{(2.7)}$ |
| | Multi. | 3 | 4 | $84.0_{(2.3)}$ | $42.3_{(8.4)}$ | $51.5_{(1.8)}$ | $68.2_{(4.6)}$ | $58.5_{(12.0)}$ | $80.2_{(2.1)}$ | $58.5_{(7.9)}$ | $63.3_{(3.0)}$ |
| **Full-context** | Single | 12 | 1 | $64.5_{(0.4)}$ | $65.8_{(0.2)}$ | $63.2_{(0.9)}$ | $67.3_{(0.5)}$ | $60.8_{(1.4)}$ | $73.5_{(0.8)}$ | $75.0_{(0.4)}$ | $67.2_{(0.1)}$ |
| | Single | 6 | 2 | $66.7_{(2.0)}$ | $66.0_{(0.4)}$ | $62.7_{(0.6)}$ | $70.5_{(2.1)}$ | $59.7_{(0.9)}$ | $77.7_{(2.2)}$ | $76.0_{(0.0)}$ | $68.5_{(0.4)}$ |
| | Single | 3 | 4 | $64.0_{(0.0)}$ | $65.8_{(0.6)}$ | $65.0_{(1.9)}$ | $67.3_{(0.2)}$ | $59.5_{(0.4)}$ | $74.2_{(1.3)}$ | $77.0_{(1.9)}$ | $67.5_{(0.8)}$ |
| | Multi. | 12 | 1 | $83.8_{(2.9)}$ | $41.0_{(10.6)}$ | $51.2_{(0.8)}$ | $68.0_{(4.5)}$ | $58.3_{(11.1)}$ | $79.0_{(3.6)}$ | $56.0_{(8.1)}$ | $62.5_{(2.8)}$ |
| | Multi. | 6 | 2 | $85.3_{(1.9)}$ | $41.2_{(10.7)}$ | $51.2_{(1.3)}$ | $67.7_{(4.5)}$ | $57.7_{(10.8)}$ | $79.2_{(3.7)}$ | $55.8_{(7.9)}$ | $62.6_{(2.6)}$ |
| | Multi. | 3 | 4 | $83.3_{(2.5)}$ | $41.7_{(11.3)}$ | $51.0_{(1.1)}$ | $68.2_{(4.7)}$ | $57.7_{(10.8)}$ | $79.0_{(3.2)}$ | $56.0_{(8.1)}$ | $62.4_{(2.8)}$ |

[9] When utilizing the full-context loss, the learning rates considered are $1e-5, 1e-6$, and $1e-7$ due to gradient summations in TINT.

Table 5: Few-shot ($k = 32$) results with different loss types, input formats, and layer-step configurations with a fixed compute budget, without calibration.

| Loss Type | Format | Layer | Step | Subj | AGNews | SST2 | CR | MR | MPQA | Amazon | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | Single | 12 | 1 | $63.3_{(0.2)}$ | $65.7_{(0.2)}$ | $71.3_{(0.6)}$ | $65.0_{(1.4)}$ | $70.7_{(0.9)}$ | $65.0_{(0.0)}$ | $76.7_{(0.2)}$ | $68.2_{(0.1)}$ |
| | Single | 6 | 2 | $63.5_{(0.0)}$ | $65.2_{(0.5)}$ | $73.3_{(1.3)}$ | $68.5_{(3.7)}$ | $71.3_{(0.2)}$ | $66.0_{(0.0)}$ | $77.5_{(0.4)}$ | $69.3_{(0.3)}$ |
| | Single | 3 | 4 | $64.2_{(0.2)}$ | $66.5_{(1.1)}$ | $73.2_{(0.6)}$ | $75.7_{(0.5)}$ | $72.0_{(0.0)}$ | $83.2_{(1.0)}$ | $78.0_{(0.4)}$ | $73.2_{(0.1)}$ |
| | Multi. | 12 | 1 | $64.5_{(7.8)}$ | $35.5_{(7.4)}$ | $56.8_{(9.7)}$ | $63.0_{(6.7)}$ | $58.7_{(8.9)}$ | $75.2_{(10.8)}$ | $62.2_{(8.3)}$ | $59.4_{(0.6)}$ |
| | Multi. | 6 | 2 | $77.7_{(7.0)}$ | $35.5_{(7.4)}$ | $57.0_{(9.9)}$ | $60.0_{(6.3)}$ | $52.3_{(2.1)}$ | $58.5_{(6.1)}$ | $55.8_{(7.9)}$ | $56.7_{(2.6)}$ |
| | Multi. | 3 | 4 | $67.5_{(11.5)}$ | $38.5_{(8.2)}$ | $55.3_{(5.2)}$ | $67.0_{(3.5)}$ | $61.0_{(8.0)}$ | $65.2_{(11.2)}$ | $62.5_{(8.9)}$ | $59.6_{(1.3)}$ |
| **Full-context** | Single | 12 | 1 | $65.5_{(1.1)}$ | $66.5_{(0.0)}$ | $70.7_{(0.2)}$ | $64.8_{(0.5)}$ | $72.0_{(1.4)}$ | $67.0_{(0.0)}$ | $76.5_{(0.0)}$ | $69.0_{(0.3)}$ |
| | Single | 6 | 2 | $64.7_{(0.6)}$ | $66.2_{(0.2)}$ | $71.2_{(0.2)}$ | $65.3_{(0.6)}$ | $71.5_{(0.4)}$ | $67.0_{(0.0)}$ | $76.7_{(0.2)}$ | $68.9_{(0.0)}$ |
| | Single | 3 | 4 | $64.2_{(0.2)}$ | $66.2_{(0.2)}$ | $71.3_{(0.2)}$ | $64.7_{(0.2)}$ | $71.0_{(0.0)}$ | $67.0_{(0.0)}$ | $76.5_{(0.0)}$ | $68.7_{(0.0)}$ |
| | Multi. | 12 | 1 | $62.2_{(7.5)}$ | $33.8_{(8.3)}$ | $52.2_{(3.1)}$ | $52.8_{(4.0)}$ | $50.8_{(1.2)}$ | $55.8_{(4.3)}$ | $55.3_{(7.2)}$ | $51.9_{(2.2)}$ |
| | Multi. | 6 | 2 | $60.0_{(5.5)}$ | $33.7_{(8.4)}$ | $50.8_{(1.2)}$ | $52.2_{(2.4)}$ | $50.2_{(0.2)}$ | $54.3_{(2.5)}$ | $55.0_{(6.7)}$ | $50.9_{(1.8)}$ |
| | Multi. | 3 | 4 | $58.7_{(4.9)}$ | $33.7_{(8.4)}$ | $50.8_{(1.2)}$ | $51.3_{(1.9)}$ | $50.0_{(0.0)}$ | $54.3_{(2.5)}$ | $55.3_{(7.2)}$ | $50.6_{(2.0)}$ |