LayerNorm vs RMSNorm: A Geometric Perspective and the Case Against Mean Subtraction

Akshat Gupta, Atahan Ozdemir, Caoqinwei Gong, Gopala Anumanchipalli
UC Berkeley
akshat.gupta@berkeley.edu

Abstract

This paper presents a novel geometric interpretation of LayerNorm and explores how LayerNorm influences the norm and orientation of hidden vectors in the representation space. We show that the definition of LayerNorm is innately linked to the uniform vector, defined as $\mathbf{1} = [1, 1, 1, 1, \cdots, 1]^T \in \mathbb{R}^d$. We then show that the standardization step in LayerNorm can be understood in three simple steps: (i) remove the component of a vector along the uniform vector, (ii) normalize the remaining vector, and (iii) scale the resultant vector by $\sqrt{d}$, where d is the dimensionality of the representation space. Finally, we compare the hidden representations of LayerNorm-based LLMs with models trained using RMSNorm and show that all LLMs naturally operate orthogonal to the uniform vector both during training and inference, that is, on average they do not have a component along the uniform vector during training or inference. This presents the first mechanistic evidence that removing the component along the uniform vector in LayerNorm is a redundant step. These results advocate for using RMSNorm over LayerNorm which is also more computationally efficient.

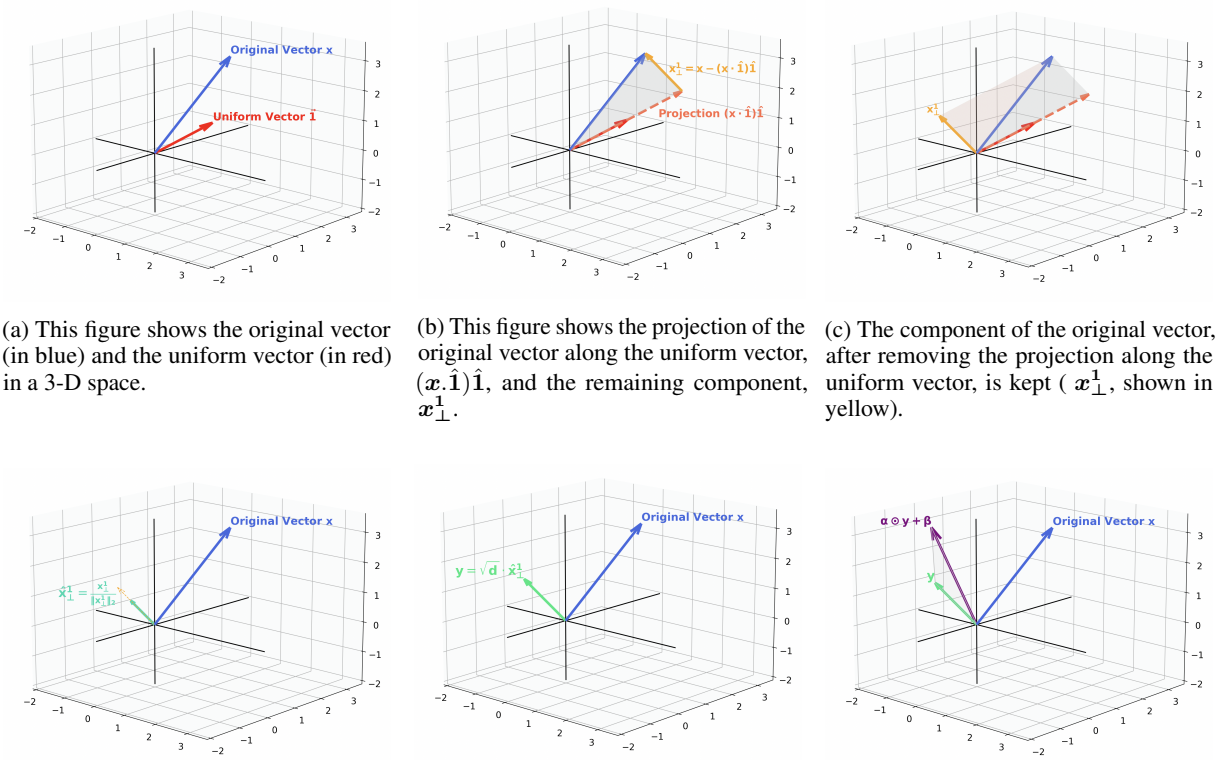
1 Introduction

The transformer architecture (Vaswani et al., 2017) has been the cornerstone of most recent advances in artificial intelligence and has rapidly become the architecture of choice in natural language processing, computer vision, speech, and various other domains. Layer normalization, a crucial yet often overlooked component of the transformer architecture, plays an integral role in stabilizing the training process in transformer-based models. Introduced by Ba et al. (2016), layer normalization standardizes the features at each layer, adjusting and scaling the activations to have zero mean and unit variance within a vector. This normalization is performed independently for each hidden vector in contrast to batch normalization (Ioffe and Szegedy, 2015), which relies on statistics (mean and variance) from a batch of data points. Layer normalization is especially effective when working with long sequences of variable lengths. While the benefits of layer normalization, such as improved training speed and better convergence, are well documented (Ba et al., 2016; Xu et al., 2019; Zhang and Sennrich, 2019; Jiang et al., 2024), its specific effects on the hidden vectors within a model and the global properties of the resulting representations remain surprisingly underexplored.

In this paper, we first discuss the global effects of layer normalization on a vector. The conventional explanation of the LayerNorm operation is usually as follows: *standardize each vector by subtracting the mean of its elements, divide by the standard deviation*. While this is an accurate procedural definition, we ask a more global question: **How does the LayerNorm operation transform a vector in representation space?**

We present a novel interpretation of LayerNorm and show that it can be understood in three steps: (i) throw away the component of the given vector along $\mathbf{1} = [1, 1, 1, 1, \cdots, 1]^T \in \mathbb{R}^d$, (ii) normalize the resultant vector; and (iii) scale the resulting vector by $\sqrt{d}$, where d is the dimensionality of the vector space. This process is illustrated in Figure 1. This operation throws away the information along $\mathbf{1} = [1, 1, 1, 1, \cdots 1]^T$, which we call the uniform vector, indicating that the information along the uniform vector may not be important.

We then present the property of "irreversibility," where we show that the information lost in the LayerNorm process cannot be recovered. These results naturally lead to discussion about the importance of information along the uniform vector, which is removed by LayerNorm irreversibly, and its comparison with RMSNorm (Zhang and Sennrich, 2019), a variant of LayerNorm that doesn't remove the component along the uniform vector



(a) This figure shows the original vector (in blue) and the uniform vector (in red) in a 3-D space.

(b) This figure shows the projection of the original vector along the uniform vector, $(\boldsymbol{x}.\hat{\mathbf{1}})\hat{\mathbf{1}}$, and the remaining component, $\boldsymbol{x}^{\mathbf{1}}_{\perp}$.

(c) The component of the original vector, after removing the projection along the uniform vector, is kept ($\boldsymbol{x}^{\mathbf{1}}_{\perp}$, shown in yellow).

(d) Normalizing perpendicular component to the uniform vector ($\boldsymbol{x}^{\mathbf{1}}_{\perp}$) to unit norm.

(e) Scale of resultant vector by $\sqrt{d}$, where d is the dimensionality of the representation space.

(f) Finally, the scale-and-shift step, which scales and shifts the resulting vector according to learnt parameters.

Figure 1: Visualization of LayerNorm operation on a random original vector

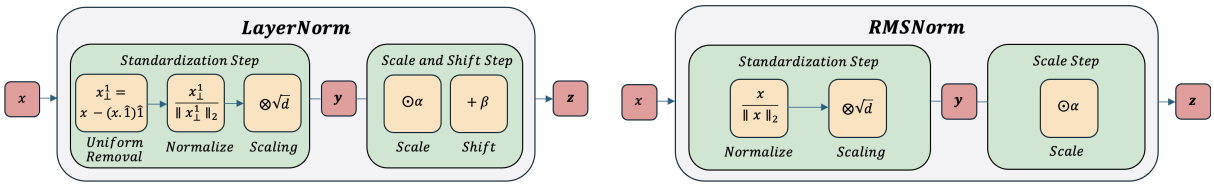


Figure 2: A diagrammatic explanation of LayerNorm and RMSNorm.

and is used to train the latest Llama models (Touvron et al., 2023a,b; Dubey et al., 2024). Figure 2 shows the difference between the two normalization methods. In high dimension representations spaces like the ones transformers operate in, two vectors chosen at random are orthogonal to each other on average. To justify removing the components of hidden vectors along the uniform vectors, the hidden representations should have non-trivial components along it.

We empirically show that LayerNorm-based models naturally produce hidden representations orthogonal to the uniform vector during both training and inference time, thus rendering the removal of the uniform vector in LayerNorm inconsequential in practice. This is true even for RMSNorm-based models, where hidden vectors on average operate orthogonal to the uniform vector. With these results, we provide both theoretical and empirical justifications for the redundancy of removing the component of hidden representations along the uniform vector, thus supporting the usage of RMSNorm over LayerNorm, which is also more computationally efficient.

2 Re-Introducing Layer Normalization

Let $\boldsymbol{x} \in \mathbb{R}^d$ represent an intermediate hidden representation in a transformer-based model, on which the LayerNorm operation is applied. Then the following steps summarize the layer normalization operation:

1. Calculate the mean, $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$, and standard deviation, $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$,

where x_i are the components of $\boldsymbol{x}$

2. Standardize the components of the hidden vector $\boldsymbol{x}$ to get $\boldsymbol{y}$, such that $y_i = \frac{x_i - \mu}{\sigma}, \forall x_i \in \boldsymbol{x}$

3. Scale and shift to get $\boldsymbol{z} = \boldsymbol{\alpha} \odot \boldsymbol{y} + \boldsymbol{\beta}$, where $\odot$ represents element-wise multiplication of two vectors, and $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ are scaling and shifting vector parameters learned during training.

In this paper, we refer to the combination of step-1 and step-2 as the *"standardization step"*, whereas step-3 is referred to as the *"scale-and-shift"* step. The removal of the mean of components from each component in step-1 is referred to as the *"mean subtraction"* step in this paper. For an overview of the computations happening within a decoder layer in modern LLMs, we refer the reader to section A.1.

While the original definition of LayerNorm gives us an operational description, it tells us very little about the global properties of the resulting vectors. For example, with the steps described above, we get very little insight into the norm or the orientation of the standardized vector ($\boldsymbol{y}$). All we know is that the components of the vector $\boldsymbol{y}$ are standardized to have zero mean and unit variance within the vector. But how is this standardized vector oriented compared to the original vector and what are its norms? Such an explanation was also absent in the original formulation of layer normalization (Ba et al., 2016).

Initial explorations into the geometry of layer normalization by Brody et al. (2023) demonstrated that $\boldsymbol{y}$ maintains a norm of $\sqrt{d}$ and is oriented orthogonal to the constant vector $\mathbf{1} = [1, 1, 1, 1, \cdots, 1]^T \in \mathbb{R}^d$, where d represents the dimensionality of the vector space of hidden vectors. Furthermore, a popular variant of LayerNorm, known as RMSNorm (Zhang and Sennrich, 2019), which omits the “mean subtraction" in the standardization step, has come out as a viable alternative to LayerNorm. In this paper, we expand upon these foundational studies by presenting a novel interpretation of the global effects of LayerNorm on a vector, which we argue provides a more intuitive and informative description of how LayerNorm modifies a hidden vector in representation space. Additionally, we analyze the representations of models using RMSNorm, showing that despite omitting the "mean subtraction step", RMSNorm produces hidden representations with similar orientations.

To understand the global effect of LayerNorm on a vector $\boldsymbol{x}$, we need to represent the standardization steps in terms of the vector $\boldsymbol{x}$ and not its components. A neat way to write the mean of the components of a vector $\boldsymbol{x}$ is:

$$\mu = \frac{1}{d}\mathbf{1}^T\boldsymbol{x} \tag{1}$$

where $\mathbf{1} = [1, 1, 1, 1, \cdots 1]^T$ such that $\mathbf{1} \in \mathbb{R}^d$, which we refer to as the *uniform vector* in this paper. Using the scalar mean value calculated above, we define the *mean vector* as

$$\boldsymbol{\mu} = \mu\mathbf{1} \tag{2}$$

where μ is a scalar mean as calculated in equation 1. Thus, the mean vector is a vector with each component equal to the mean of the components of vector $\boldsymbol{x}$. Using this, we can also rewrite the standard deviation of the components of $\boldsymbol{x}$ as:

$$\sigma = \sqrt{\frac{1}{d}(\boldsymbol{x} - \boldsymbol{\mu})^T(\boldsymbol{x} - \boldsymbol{\mu})} \tag{3}$$

Once we do this, we can now write the standardize step in the LayerNorm algorithm (step-2) in vector form as follows:

$$\boldsymbol{y} = \frac{1}{\sigma}(\boldsymbol{x} - \boldsymbol{\mu}) \tag{4}$$

In the above equation, we are assuming that the standard deviation is non-zero. As can be seen in equation 3, the standard deviation is zero only when $\boldsymbol{x} = \boldsymbol{\mu}$, which happens when all components of the hidden vector are equal. In practice, the authors of LayerNorm add an error term in the denominator to prevent this from happening. In the discussion that follows, we will assume that $\boldsymbol{x} \neq \boldsymbol{\mu}$, which is the same as saying that all components of $\boldsymbol{x}$ are not equal. This assumption does not lead to any loss of generality, as will be evident in later discussions. If the standard deviation is zero or $\boldsymbol{x} = \boldsymbol{\mu}$, then LayerNorm outputs a vector with all its components equal to zero.

2.1 The Uniform Vector and the Mean Vector

The vector definition of layer normalization in equation 4 requires us to define two new vectors: the *uniform vector* and the *mean vector*. Since these are non-standard vectors, it is important to understand the properties of these vectors.

The *uniform vector* or $\mathbf{1} = [1, 1, 1, 1, \cdots 1]^T$, is called so because all its components are equal

or uniform and set to 1. This vector plays a very important role in the formulation and understanding of layer normalization. An important thing to note here is the norm of the uniform vector: $\|\mathbf{1}\|_2 = \sqrt{\mathbf{1}^T\mathbf{1}} = \sqrt{d}$.

The *mean vector* in the context of LayerNorm is defined in a non-traditional manner. Traditionally, a mean vector is the sum of a few vectors. But in our definition, the mean vector, $\boldsymbol{\mu} = \mu\mathbf{1}$, is a scaled version of the uniform vector. It is scaled by a value that is the mean or the average of all the components of the hidden vector and is oriented in the direction of the uniform vector. An interesting property of the mean vector is its norm. Let's define θ_{x1} as the angle between vectors $\boldsymbol{x}$ and the uniform vector. Then taking the L2 norm of the mean vector:

$$\begin{aligned}\|\boldsymbol{\mu}\|_2 = \|\mu\mathbf{1}\|_2 = \mu\|\mathbf{1}\|_2 = \left(\frac{1}{d}\mathbf{1}^T\boldsymbol{x}\right)\|\mathbf{1}\|_2 \\ = \frac{1}{d}\|\mathbf{1}\|_2^2\|\boldsymbol{x}\|_2\cos\theta_{x1} = \|\boldsymbol{x}\|_2\cos\theta_{x1}\end{aligned} \tag{5}$$

Here we replace the formula for the mean from equation 1, expand the inner product between the uniform vector and $\boldsymbol{x}$ in terms of their norms and the angle between them, and use the fact that $\|\mathbf{1}\|_2 = \sqrt{d}$.

The norm of the *mean vector* gives us very important insights about what's going on in the layer normalization process. Equation 5 shows that the norm of the mean vector is nothing but **the projection of the underlying vector $\boldsymbol{x}$ along the uniform vector**. By definition, the mean vector, $\boldsymbol{\mu} = \mu\mathbf{1}$, is oriented along the direction of the uniform vector. In other words, given that θ is the angle between vectors $\boldsymbol{x}$ and the uniform vector,

$$\|\boldsymbol{\mu}\|_2 = \|\boldsymbol{x}\|_2\cos\theta_{x1} = \frac{\boldsymbol{x}\cdot\mathbf{1}}{\|\mathbf{1}\|_2} = \boldsymbol{x}\cdot\hat{\mathbf{1}} \tag{6}$$

and

$$\boldsymbol{\mu} = (\boldsymbol{x}.\hat{\mathbf{1}})\hat{\mathbf{1}} \tag{7}$$

Note that $\hat{\mathbf{1}}$ is the unit vector corresponding to the uniform vector $\mathbf{1}$. To the best of our knowledge, the geometrical meaning of the mean vector has not been discussed in literature before our work. In the next section, we incorporate this information to explain the LayerNorm process, giving a very intuitive and informative description of the process.

2.2 Explanation of Layer Normalization

If we go through the steps of layer normalization as discussed in the beginning of section 2, the component-wise subtraction of the mean of all components of a vector can now be written completely in terms of the vector $\boldsymbol{x}$. We define $\boldsymbol{x}_\perp^{\mathbf{1}}$ as the component of $\boldsymbol{x}$ orthogonal to the uniform vector as follows:

$$\boldsymbol{x}_\perp^{\mathbf{1}} = \boldsymbol{x} - \boldsymbol{\mu} = \boldsymbol{x} - (\boldsymbol{x}.\hat{\mathbf{1}})\hat{\mathbf{1}} \tag{8}$$

Thus, subtracting the mean of the components of a vector is the same as the removal of the projection of the vector along the uniform vector. The complete standardization step of the layer normalization operation can now be written as:

$$\boldsymbol{y} = \sqrt{d}\frac{\boldsymbol{x}_\perp^{\mathbf{1}}}{\|\boldsymbol{x}_\perp^{\mathbf{1}}\|_2} \tag{9}$$

where d is the dimensionality of the vector being normalized. This shows that **layer normalization can be simply defined as the normalization of the component of a vector orthogonal to the uniform vector, accompanied by a scaling factor**. With this, we present the simple and elegant definition of the layer normalization process with a deep geometric meaning. The most intuitive recipe of the layer normalization process is shown in Algorithm 1.

Algorithm 1 : The Standardization Step in Layer Normalization

1: Throw away the component of a vector along the uniform vector, $\mathbf{1} = [1, 1, 1, 1, \cdots 1]^T$
2: Normalize the remaining vector
3: Scale the resulting vector by $\sqrt{d}$, where d is the dimensionality of the vector space

Equation 9 also shows that the norm of the standardized vector is $\sqrt{d}$ and is oriented orthogonal to the direction of the uniform vector, which means it exists in a subspace orthogonal to the uniform vector.

2.3 The Irreversibility of Layer Normalization

The idea of reversibility is discussed briefly while introducing batch normalization (Ioffe and Szegedy, 2015), which normalizes each feature dimension in the input vector $\boldsymbol{x}$ independently by calculating the mean and standard deviation statistics over a large sample of such vectors. Specifically,

let $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots \boldsymbol{x}^b$ be a set of b vectors in the training set, each vector of dimension d. Then, both the batch normalization and layer normalization processes for each component can be represented in the following two-step process:

$$y_i = \frac{x_i - \mathbb{E}[x_i]}{\sqrt{Var(x_i)}} \text{ (standardization)} \quad (10)$$

$$z_i = \alpha_i x_i + \beta_i \text{ (scale-and-shift)} \quad (11)$$

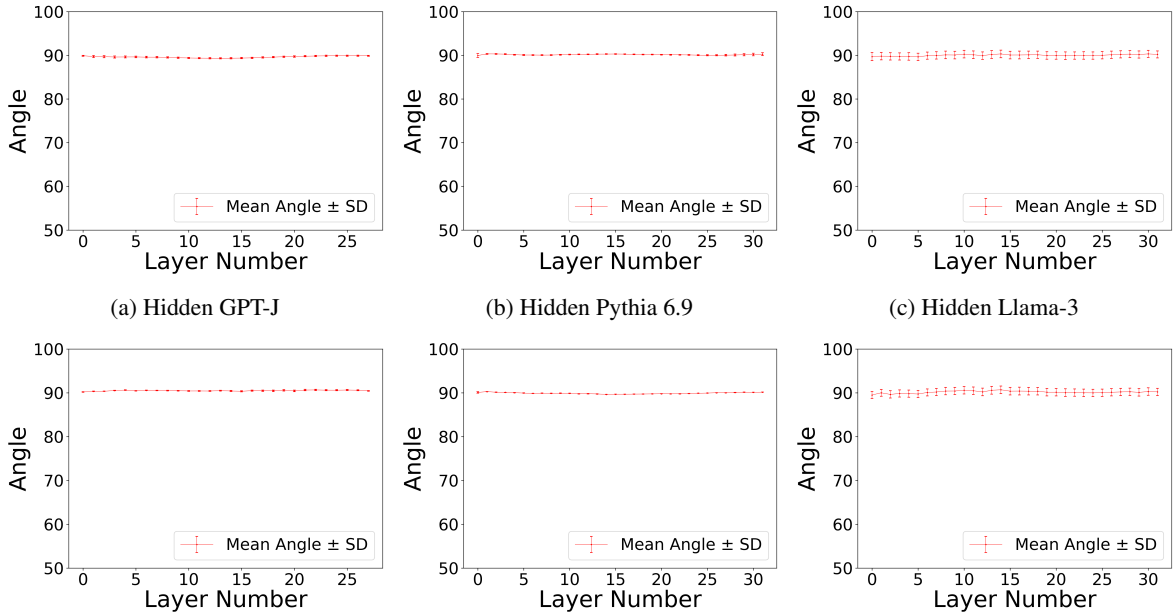
The key difference between batch normalization and layer normalization lies in how the expectation values and variance are calculated for each component. In the case of batch normalization, the mean and variance for each dimension are calculated over the training set. This means that $\mathbb{E}[x_i]$ and $Var(x_i)$ are the same for each vector $\boldsymbol{x}^j \in \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots \boldsymbol{x}^b\}$, but different for each component of the vector. In total, for a d-dimensional hidden vector space, there are $2d$ statistics in batch normalization, d for the mean and d for variance. Due to this, the information lost during the standardization step in batch normalization, if important, can be recovered in the scale-and-shift step by simply learning $\alpha_i = \sqrt{Var(x_i)}$ and $\beta_i = \mathbb{E}[x_i]$, since we have $2d$ learnable parameters in batch normalization. The scale-and-shift step was a very conscious design choice of the inventors of batch normalization to allow for batch normalization to represent an identity transformation if the original hidden representations were optimal for the network.

A subtle but very important difference with layer normalization is how the expectation and variances are calculated for each component. For layer normalization, these statistics are calculated for each vector independently. $\mathbb{E}[x_i]$ and $Var(x_i)$ in layer normalization are the same for each component x_i within a vector but are different for different vectors. This means that two variables are created in LayerNorm per hidden vector. As LLMs will normalize way more than d vectors throughout the course of their training, thus creating more than $2d$ statistics, the number of statistic variables used in LayerNorm is very high. Thus, the information lost during layer normalization cannot be recovered by learning just $2d$ components of the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ learnable parameters. This shows that **information once lost during layer normalization cannot be recovered during the scale-and-shift step, making the layer normalization process irreversible**.

3 LayerNorm versus RMSNorm

Above analysis shows a critical aspect of LayerNorm - its ability to orient incoming hidden vectors orthogonal to the uniform vector. In section 2.2, we see that the definition of LayerNorm is innately linked with the uniform vector, possibly unintended by the original authors. Since the information along the uniform vector is being removed irreversibly during layer normalization (section 2.3), the layer normalization process implicitly assumes that the information along the uniform vector is either not important or that the model should not store information along that direction. Since the "mean subtraction" step in LayerNorm removes the component along the arbitrarily chosen direction of the uniform vector, the need for such a step would be justified if the hidden representations created by the model had unnaturally significant components along this vector. However, there seems to be no justification given in the literature for the mean subtraction step, including the original paper (Ba et al., 2016). RMSNorm, on the other hand, does not perform this step and simply normalizes the existing vector (Zhang and Sennrich, 2019). The Llama model series (Touvron et al., 2023a,b) is the most prominent family of LLMs that use RMSNorm. The simple fact that Llama models achieve state-of-the-art results across multiple measures (Dubey et al., 2024) shows that using RMSNorm instead of LayerNorm does not hurt performance. This provides a strong motivation to explore the need for the "mean subtraction" step in LayerNorm.

Two randomly chosen vectors in high-dimensional spaces are nearly orthogonal to each other with a very small spread. While the choice of uniform vector as one of the two vectors may seem random, the hidden representations generated by a transformer during a computation belong to a specific distribution which lie in specific regions in the representation space. Thus, the uniform vector and vectors belonging to the distribution of hidden representations may or not be orthogonal. Thus, we ask the question - do intermediate hidden representations have a non-trivial component along the uniform vector that justifies its removal? If this is true, it may present a case for removing the component along the uniform vector. We answer this question empirically.



(a) Hidden GPT-J (b) Hidden Pythia 6.9 (c) Hidden Llama-3

(d) Post-LN1 GPTJ (e) Post-LN1 Pythia 6.9 (f) Post-LN1 Llama-3

Figure 3: Error bars of angles (in degrees) between Hidden vectors (a-c) and post-normalization vectors (d-f) with the *uniform vector* for GPT-J, Pythia 6.9, Llama-3 for all layers.

Model Name	Model Dim (d)	Num Params	Num Layers	Norm. Type
GPT-2 XL	1600	1.5B	48	Layer
GPT-Neo 1.3B	2048	1.3B	24	Layer
Pythia-1.4B	2048	1.4B	24	Layer
GPT-J 6B	4096	6.0B	28	Layer
Pythia-6.9B	4096	6.9B	32	Layer
Llama-2-7B	4096	7.0B	32	RMS
Llama-3-8B	4096	8.0B	32	RMS

Table 1: List of models used in experiments.

3.1 Experimental Setup

We study the angle between the hidden representations generated within a model and the uniform vector both during training and inference. To do so, we pass one million tokens from Wikipedia articles through various models and study how the hidden representations corresponding to these vectors align with the uniform vector.

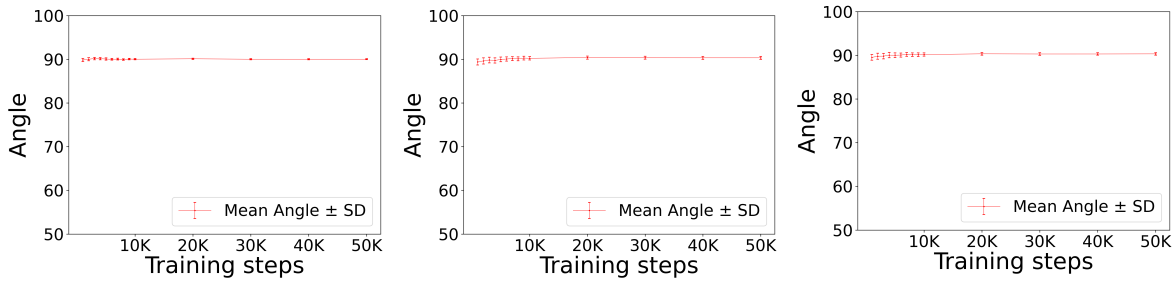
Methods and Models We study the impact of LayerNorm on the hidden representations for 7 decoder-only LLMs across two size categories. The models used are listed in Table 1. GPT2XL, GPT-Neo-1.3B, and Pythia-1.4B represent the small LLM size category, whereas GPT-J-6B, Pythia-6.9B, Llama-2-7B, and Llama-3-8B represent the medium LLM size category. We pass one million tokens from Wikipedia articles through each model and capture the hidden representations for all tokens before and after each normalization layer. This is done for each layer inside the model, which creates many terabytes of data to be analyzed.

3.2 Orthogonality to Uniform Vector at Inference time

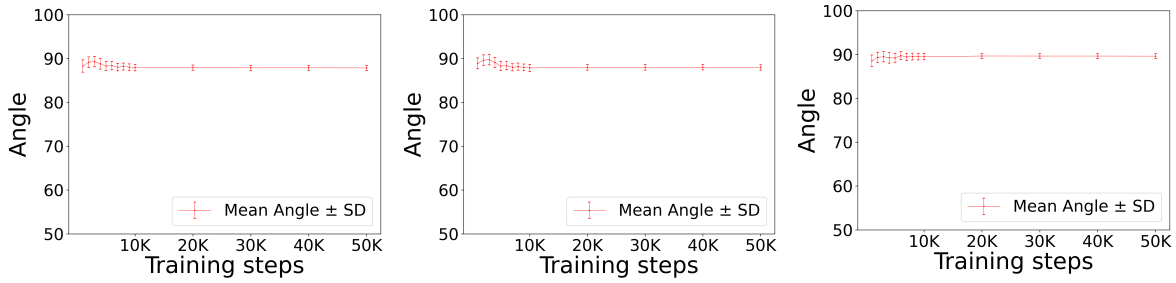
To answer the question of alignment of the uniform vector at inference time, we measure the angle between the hidden vector and the uniform vector just before and after the normalization operations for LayerNorm and RMSNorm-based models. To justify the mean subtraction step, the following scenarios should be true:

- Scenario-1: In LayerNorm-based models, the intermediate hidden vectors have a large component along the uniform vector pre-normalization, which gets removed post-normalization.
- Scenario-2: In RMSNorm-based models, which are trained without the mean subtraction step, the hidden vectors consistently have a large component along the uniform vector.

More specifically, we find the angle between the hidden vectors h^{l-1} (equation 12) with the uniform vectors, which represent the hidden vectors before normaliztion. And then we also find the angle between the post normalization vectors f^l (equation 12), which represents the activation vector post normalization.



(a) Angle with the *uniform vector* for 1B with LayerNorm (b) Angle with random vector $\tilde{\mathbf{a}}$ for 1B with LayerNorm (c) Angle with random vector $\tilde{\mathbf{b}}$ for 1B with LayerNorm



(d) Angle with the *uniform vector* for 1B with RMSNorm (e) Angle with random vector $\tilde{\mathbf{a}}$ for 1B with RMSNorm (f) Angle with random vector $\tilde{\mathbf{b}}$ for 1B with RMSNorm

Figure 4: Error bars of angles (in degrees) between Hidden vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for 1B with LayerNorm (a-c) and RMSNorm (d-f) for a randomly selected layer(Layer 10). The results are independent of the choice of layers.

The results are shown in Figure 3 for GPT-J (6B), Pythia (6.9B), and Llama-3 (8B) and Figure 6 for the remaining models. We see that for all models in Figure 3, the angle between the hidden vectors and the uniform vectors is 90 degrees on average even before normalization, with a very small spread. This remains true post-normalization as well for all three models. This result is surprising because the hidden representations for LayerNorm-based models (GPT-J and Pythia) are themselves orthogonal to the uniform vector even before the LayerNorm operation. This is true for 4 out of the 5 LayerNorm-based models studied in this paper except for GPT2-XL (Figure 6 in the appendix). This shows that the “mean subtracting" operation of LayerNorm is redundant during inference, as there is nothing to remove.

For RMSNorm-based LLMs, the result for Llama-3 (8B) in Figure 3 represents scenraio -2. We see that even for Llama-3 (8B) trained using RMSNorm, the hidden representations before normalization are also on average orthogonal to the uniform vector with a very small spread. After RMSNorm, the hidden representations continue to be orthogonal to the uniform vector. This is also true for Llama-2 (7B) (Figure 6 in the appendix). This shows that even without the “mean subtraction" step during training as in RMSNorm, the hidden vectors operate orthogonal to the uniform vector, underscoring the redundancy of the “mean subtraction" step.

3.3 Orthogonality to Uniform Vector during Training

Next, we study whether hidden vectors are orthogonal to the uniform vector during pretraining. While we see this orthogonality at inference time, we want to study if this orthogonality is a by product of model training. To do so, we train a GPT2 architecture with identical data and training conditions, with the only difference being the use of LayerNorm versus RMSNorm, and compare the angle between intermediate hidden representations at each layer and the uniform vector. We use the OpenWebtext dataset to train models with different sizes, 70M, 160M, 410M and 1B. (Training details can be found in APPENDIX A.2)

We pass one million tokens from Wikipedia articles through multiple checkpoints during the training process of models to capture hidden representations of all tokens before each normalization layer, following the previous evaluation method. The results are shown in Figure 4. We present the results for the 1B model in the main paper while the rest

of the results can be found in the appendix. The first row of figures presents the angles of the hidden representations with the uniform vectors and two randomly chosen vectors $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ for the 1B model trained using LayerNorm. The second row shows the same for RMSNorm models. For LayerNorm, angles between hidden vectors and the uniform vector before normalization are 90 degrees on average for all checkpoints, which indicates that mean subtraction is dispensable during the pre-training step as well. The same pattern can be observed for RMSNorm. Even without mean subtraction, hidden vectors are already orthogonal to the uniform vector during training.

Furthermore, we aim to investigate whether the randomly chosen direction of the uniform vector in LayerNorm exhibits any distinctive characteristics. To evaluate this, we calculate angles between hidden vectors and with 2 randomly chosen vectors in the representation space. Figure 4 shows the results for 1B with LayerNorm and RMSNorm, where random vector $\tilde{\mathbf{a}}$ is arbitrarily chosen with the same norm as the *uniform vector* and random vector $\tilde{\mathbf{b}}$ is arbitrarily chosen with different norm from the *uniform vector*. As can be seen in Figure 4, angles with arbitrarily chosen vectors remain at an average of 90 degrees, exhibiting no significant difference compared to those with the uniform vector. These experiments clearly show that the uniform vector behaves quite similar to arbitrarily chosen vectors in the representation space. The reiterates the needlessness of removing the component along the uniform vector as done during LayerNorm.

With this, we provide the first mechanistic evidence that the **"mean subtraction" step in LayerNorm is dispensable**. This line of investigation was made possible by our novel geometrical interpretation of LayerNorm (section 2.2), which presented the global implications of the mean subtraction step in LayerNorm - removing the component of intermediate hidden representations along the uniform vector. Due to these results, we advocate for using RMSNorm over LayerNorm, which is also computationally more efficient and leads to comparable downstream performance.

4 Conclusion

We present a detailed theoretical and empirical analysis of layer normalization on hidden vectors in representation space, with a focus on the global effects of the LayerNorm operation on a vector. We first show that the LayerNorm operation can be understood in three simple steps: removing the component of a vector along the uniform vector ($\mathbf{1} = [1, 1, 1, 1, \cdots, 1]^T \in \mathbb{R}^d$), normalizing the remaining vector, and scaling the resultant vector by $\sqrt{d}$. We then show that LayerNorm is an irreversible process—information along the uniform vector is removed during LayerNorm and cannot be recovered using the learnable parameters available in the formulation. This is in contrast with BatchNorm, where the network has the option of learning an identity operation. Finally, we show that the "mean subtraction" operation in LayerNorm is dispensable both during inference and for training.

5 Limitations

This study provides valuable insights into the theoretical and empirical effects of layer normalization. Although the study empirically tests 7 models across two sizes and multiple language model families, the size of the models tested is restricted to 8 billion parameters. Since we wanted our results to have a large enough sample size, we chose to store hidden representations of 1 million tokens. Even for a small model like GPT2-XL, this requires 2TB of data, and for larger models like Llama3-8B, it requires storing approximately 4TB of data. Storing such large amounts of hidden representations pushed the capacity of the computational resources available to us. Because of this restriction, we leave testing our findings for larger models to future works and groups that have access to larger amounts of computing.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450.

Shaked Brody, Uri Alon, and Eran Yahav. 2023. On the expressivity role of layernorm in transformers' attention. arXiv preprint arXiv:2305.02582.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by

reducing internal covariate shift. In International conference on machine learning, pages 448–456. pmlr.

Zixuan Jiang, Jiaqi Gu, Hanqing Zhu, and David Pan. 2024. Pre-rmsnorm and pre-crmsnorm transformers: equivalent and efficient pre-ln transformers. Advances in Neural Information Processing Systems, 36.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023a. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023b. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288.

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

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. 2019. Understanding and improving layer normalization. Advances in neural information processing systems, 32.

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

A Appendix

A.1 Computations in Decoder-only LLMs

In this paper, we study the hidden representations of modern decoder-only large language models (LLMs). Let h^l represent the intermediate hidden state vectors between each decoder layer. As depicted in Figure 5, the computations within a layer of most decoder-only LLMs proceed as follows:

$$f^l = \texttt{LN1}(h^{l-1}) \tag{12}$$

$$a^l = \texttt{Att}(f^l) \tag{13}$$

$$g^l = \texttt{LN2}(h^{l-1} + a^l) \tag{14}$$

$$m^l = W^l_{proj}\sigma(W^l_{fc}g^l + b^l_{fc}) + b_{proj} \tag{15}$$

$$h^l = h^{l-1} + a^l + m^l \tag{16}$$

The intermediate hidden vectors between each layer, h^l, are also sometimes called the *residual stream*. Let LN1 represent the first LayerNorm function that acts just before the attention module and

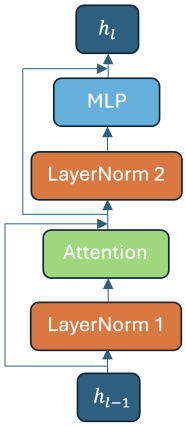


Figure 5: A high-level diagram representing the computation within one decoder block of an LLM.

LN2 represent the second LayerNorm just before the MLP module. We abstract out the computations of the attention and MLP modules to focus on the layer normalization blocks.

As the vectors computed in the attention and MLP modules get added to the residual stream at each layer, the residual stream represents a summation of an increasing number of vectors. A non-recursive formula for the residual stream depicts this clearly:

$$h^l = h^0 + \sum_{i=0}^{i=l} a^i + \sum_{i=0}^{i=l} m^i \tag{17}$$

The residual stream thus represents a continuous summation of vectors computed at the attention and MLP modules in each layer with the incoming residual stream. This leads to an increasing norm of the residual stream, thus necessitating a normalization operation.

A.2 Training Configurations

For all models, we use OpenWebText(Gokaslan and Cohen, 2019) to train with AdamW optimizer, which is an opensource recreation of the WebText corpus and extensively utilized for GPT2 pretraining. For 70M and 160M, the total training steps is 100000 and checkpoints are saved at 1K, 2K,..., 10K, 20K,..., 100K. For 410M and 1B, the total training steps is 50000 and checkpoints are saved at 1K, 2K,..., 10K, 20K,..., 50K. The max sequence length is set to 1024 and batch size is set to 64. The learning rate schedule integrates a linear warm-up phase, followed by a cosine decay scheduler with a minimum of 10% the initial learning rate. For 70M and 160M, we use 4 NVIDIA A6000 GPUs with 48GB of GPU memory. For 410M, we use 2

Model Size	Layers	Embed Dim	Heads	Learning Rate
70M	6	512	8	10.0×10^{-4}
160M	12	768	12	6.0×10^{-4}
410M	24	1024	16	3.0×10^{-4}
1B	16	2048	8	3.0×10^{-4}

Table 2: Model Configurations

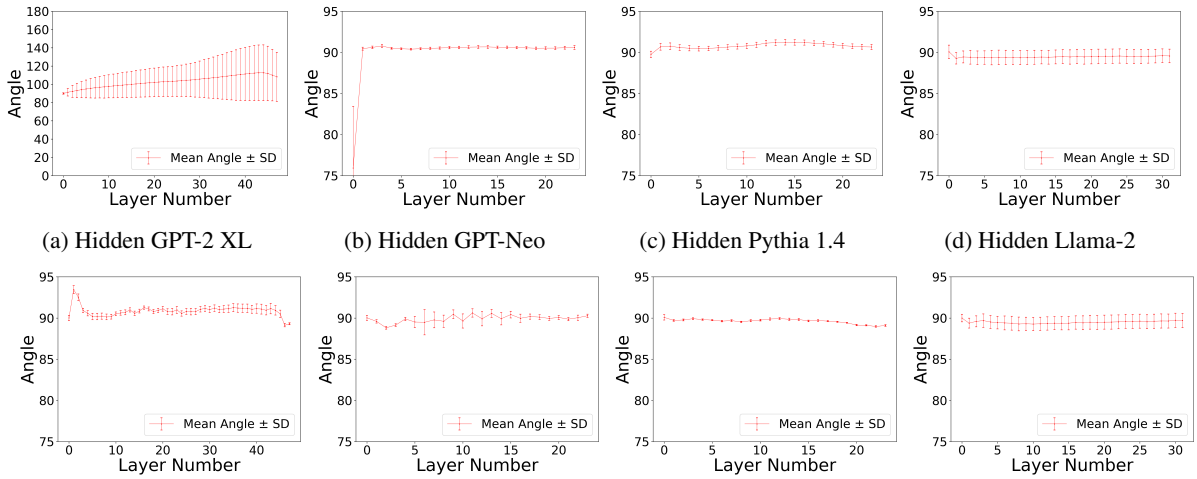
NVIDIA A100 GPUs with 80GB of GPU memory. For 1B, we use 4 NVIDIA A100 GPUs with 80GB of GPU memory. Model configurations including number of layers, attention heads, embedding dimensions and initial learning rate can be found in Table 2.

A.3 Computational Resource for experiments at inference time

All experiments at inference time were conducted using NVIDIA A6000 GPUs with 48GB of GPU memory. This setup provided the necessary computational power to handle large-scale models and extensive data processing required for our study.

A.4 Supplementary figures

Figure 7, 8, 9, 10, 11, 12 and 13 show the error bars of angles between hidden vectors and post-normalization vectors with the *uniform vector* and random vector a, b for all models listed in Table 1, where random vector a is arbitrarily chosen with the same norm as the *uniform vector* and random vector b is arbitrarily chosen with different norm from the *uniform vector*. Figure 14, 15 and 16 show the error bars of angles between hidden vectors with the uniform vector and random vector a,b for all layers in 70M with LayerNorm during training. With the same meaning, Figure 17, 18 and 19 are for 70M with RMSNorm, Figure 20, 21 and 22 are for 160M with LayerNorm, Figure 23, 24 and 25 are for 160M with RMSNorm, Figure 26, 27 and 28 are for 410M with LayerNorm, Figure 29, 30 and 31 are for 410M with RMSNorm, Figure 32, 33 and 34 are for 1B with LayerNorm and Figure 35, 36 and 37 are for 1B with RMSNorm.



(a) Hidden GPT-2 XL (b) Hidden GPT-Neo (c) Hidden Pythia 1.4 (d) Hidden Llama-2

(e) Post-LN1 GPT-2 XL (f) Post-LN1 GPT-Neo (g) Post-LN1 Pythia 1.4 (h) Post-LN1 Llama-2

Figure 6: Error bars of angles (in degrees) between Hidden vectors (a-d) and post-normalization vectors (e-h) with the uniform vector for GPT-2 XL, GPT-Neo, Pythia 1.4, and Llama-2 for all layers

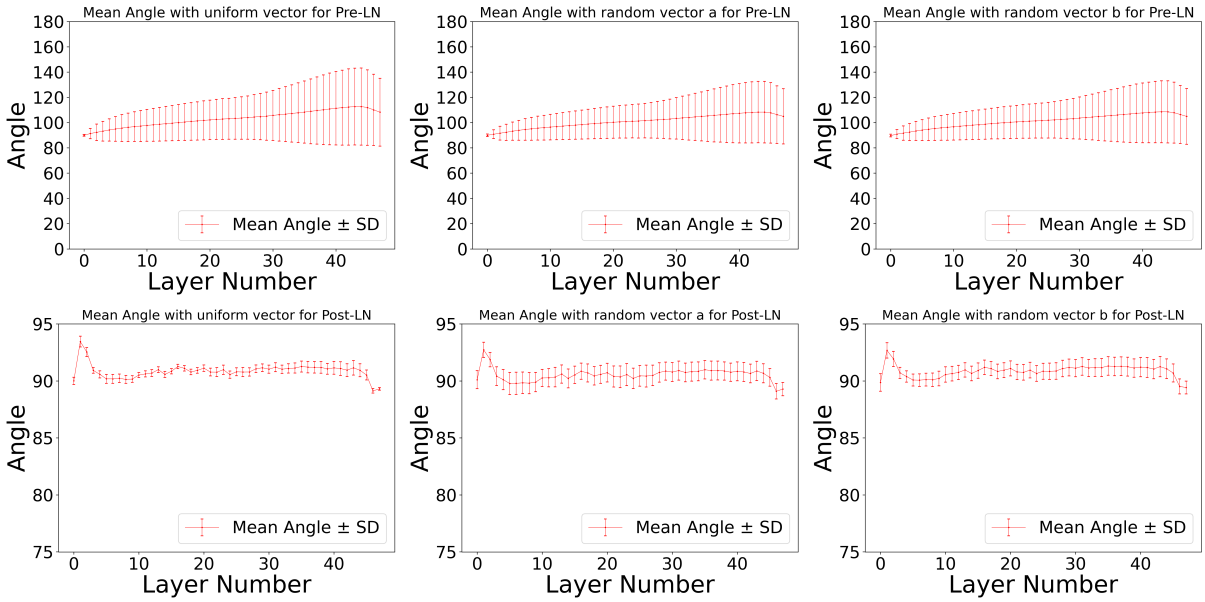


Figure 7: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in GPT-2 XL

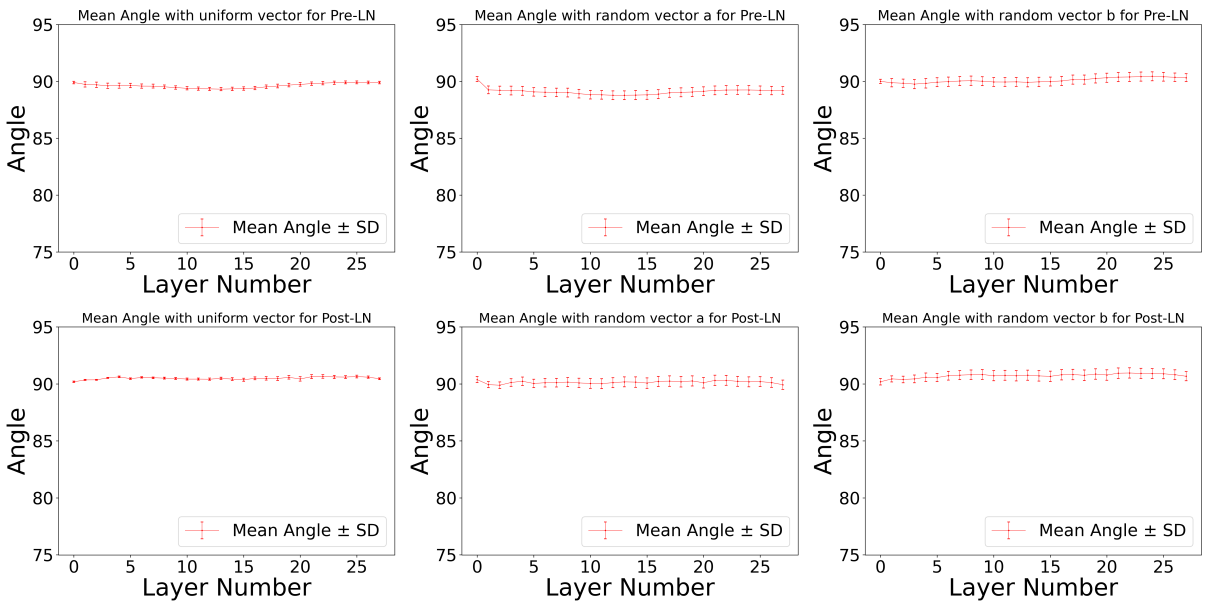


Figure 8: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in GPT-J

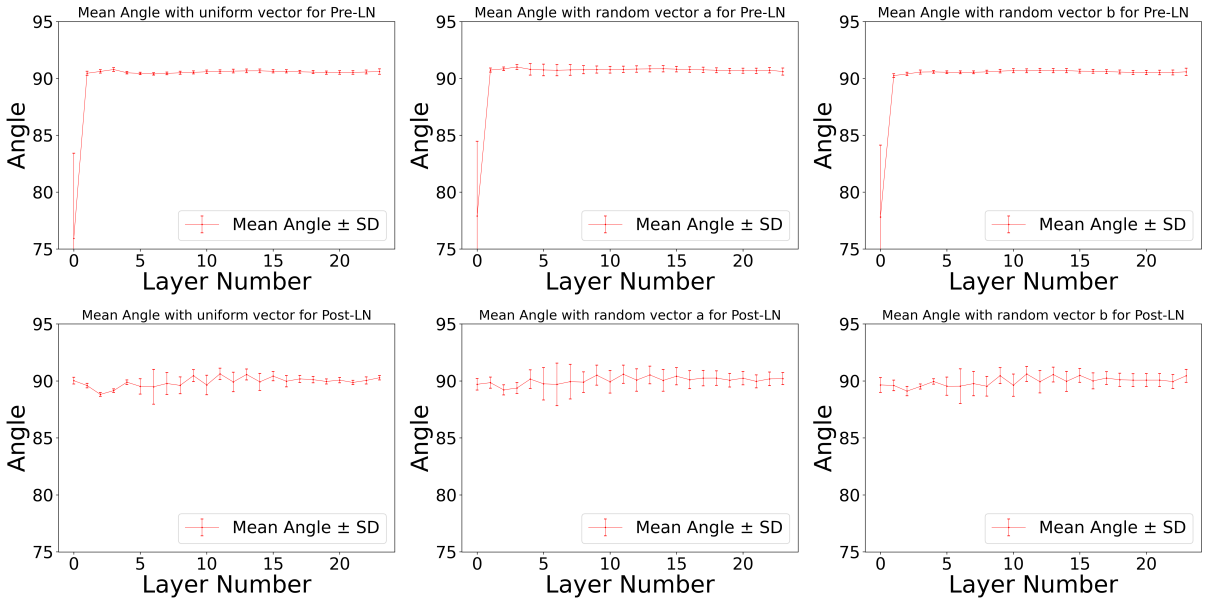


Figure 9: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in GPT-Neo

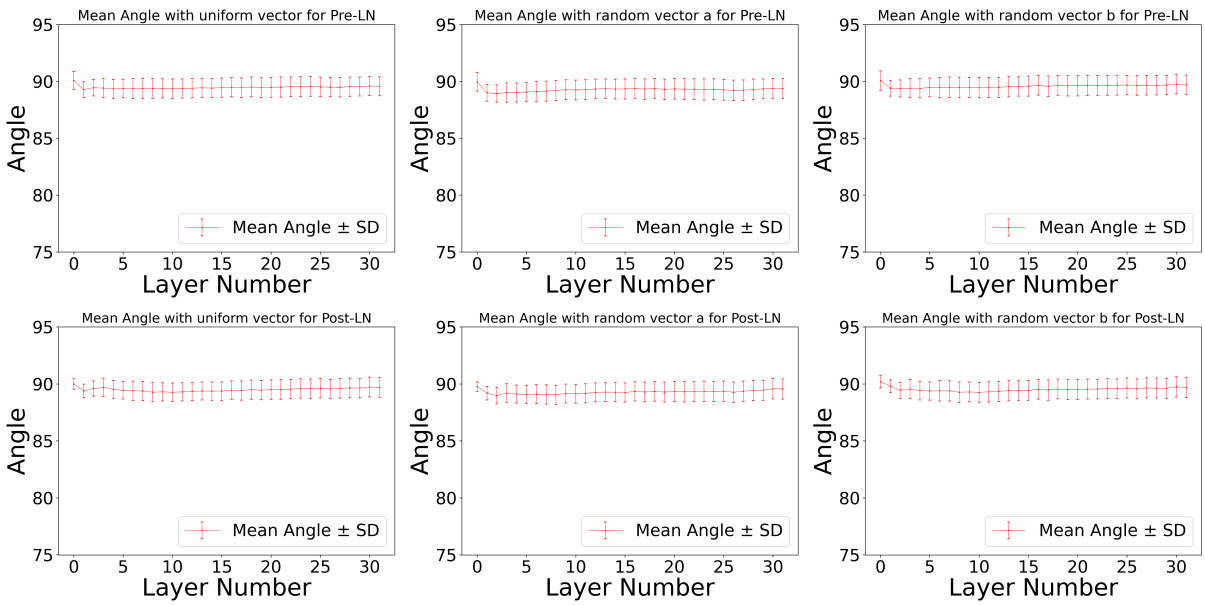


Figure 10: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in Llama-2

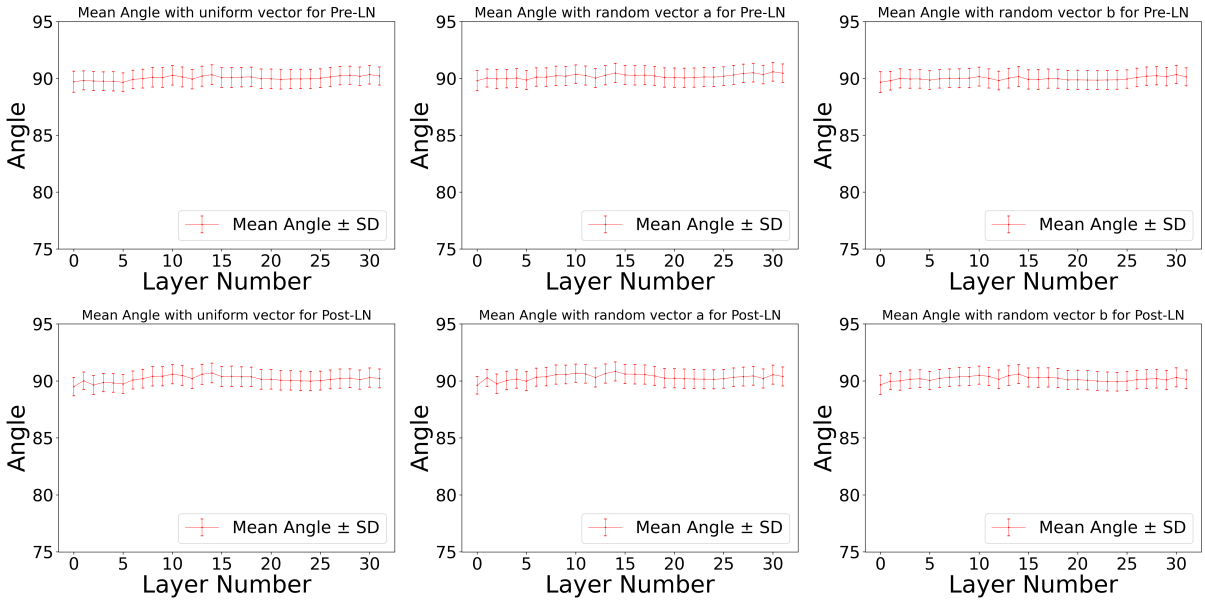


Figure 11: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in Llama-3

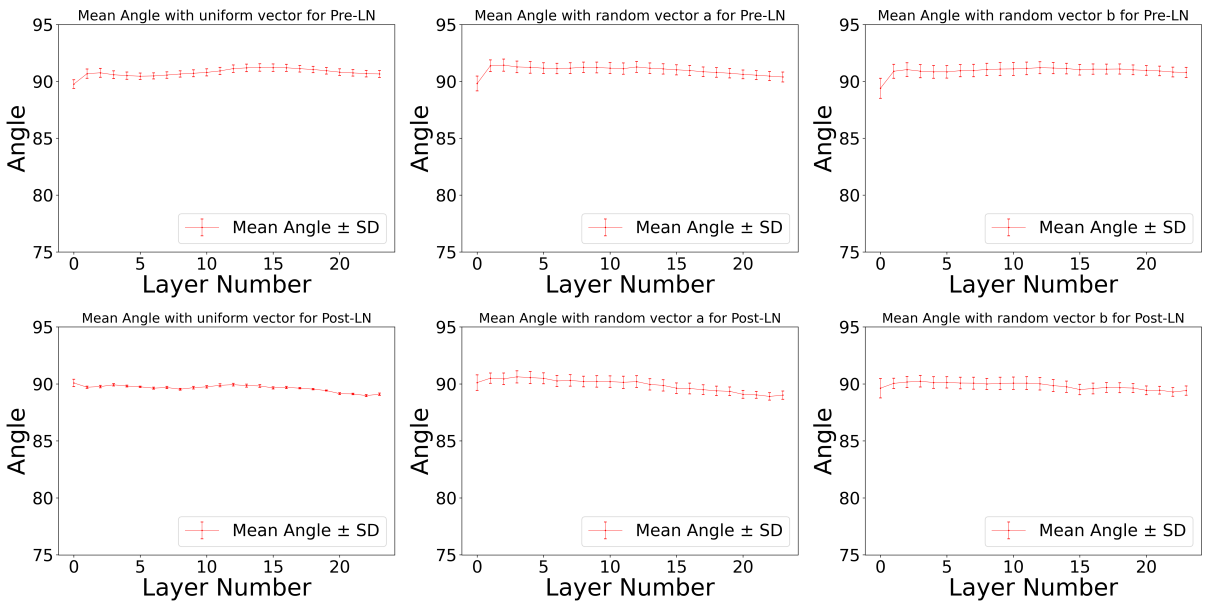


Figure 12: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in Pythia-1.4B

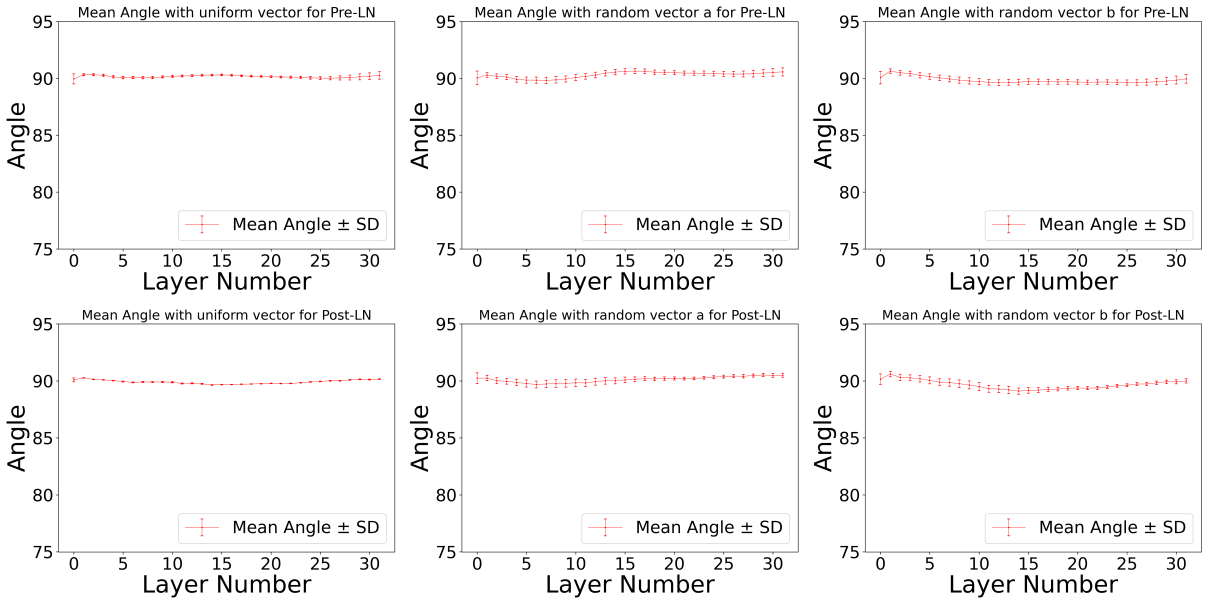


Figure 13: Error bars of angles (in degrees) between Hidden vectors and post-normalization vectors with the *uniform vector* and random vector $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$ for all layers in Pythia-6.9B

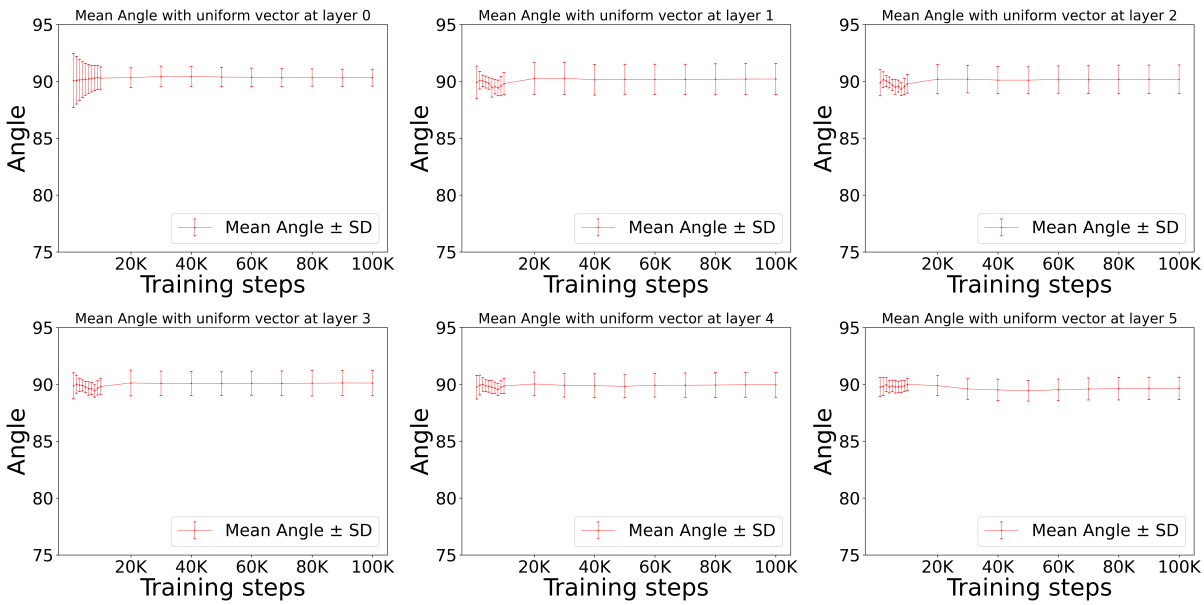


Figure 14: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 70M with LayerNorm during training

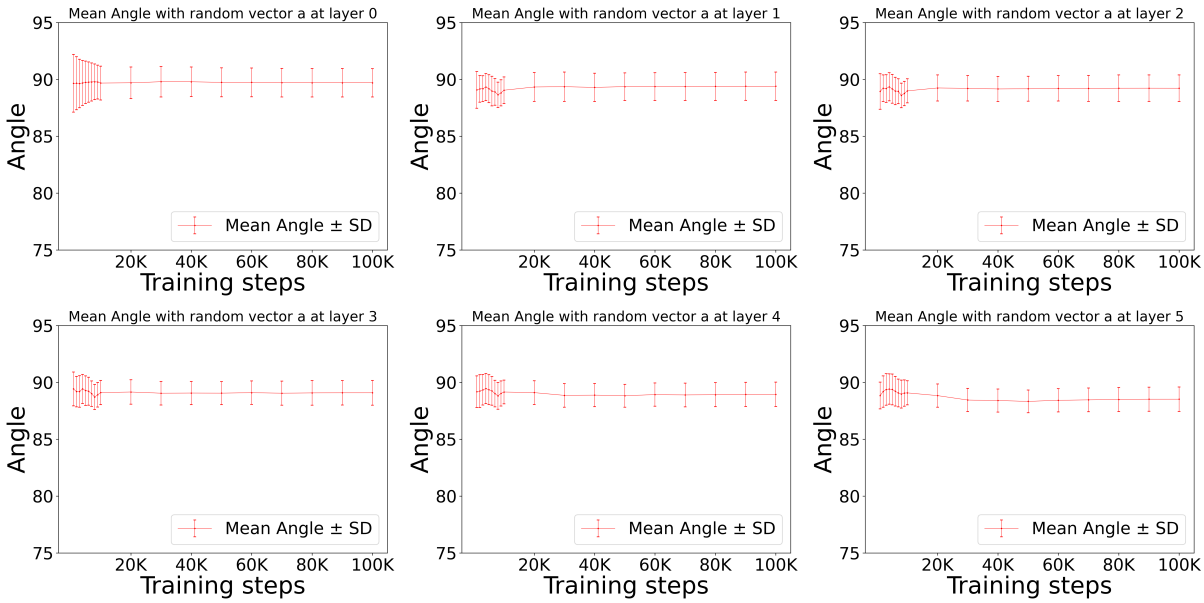


Figure 15: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 70M with LayerNorm during training

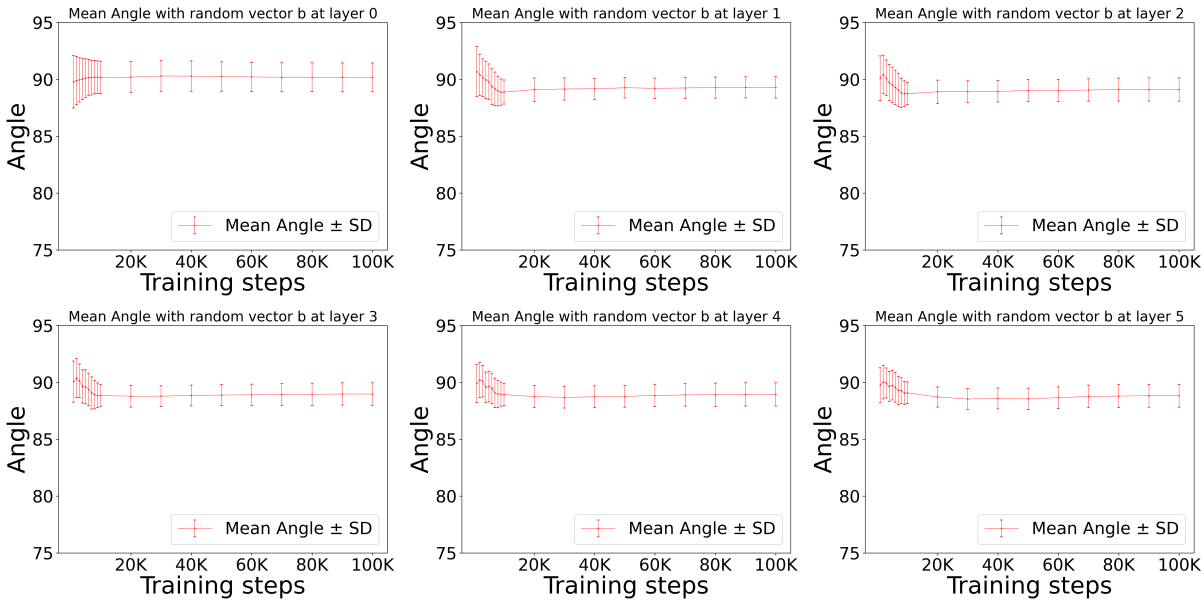


Figure 16: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 70M with LayerNorm during training

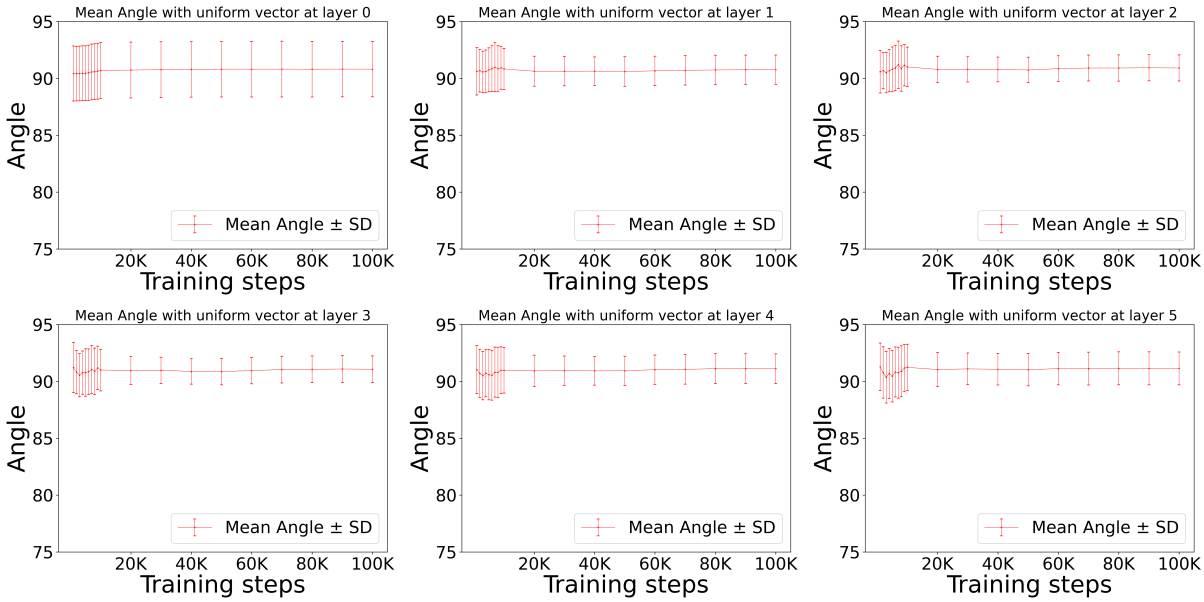


Figure 17: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 70M with RMSNorm during training

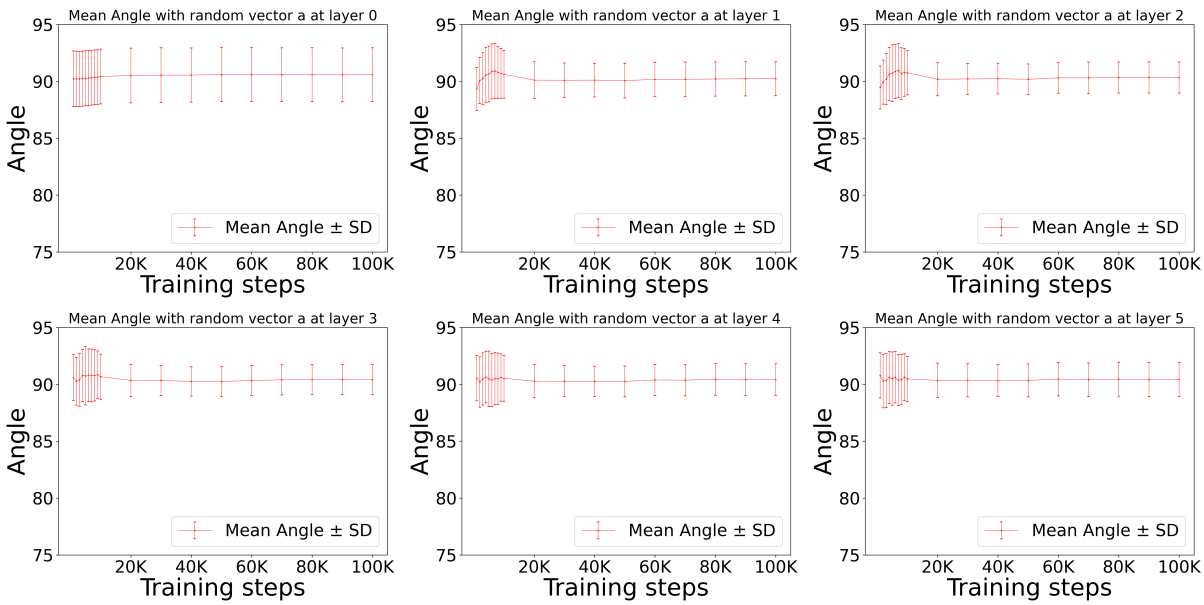


Figure 18: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 70M with RMSNorm during training

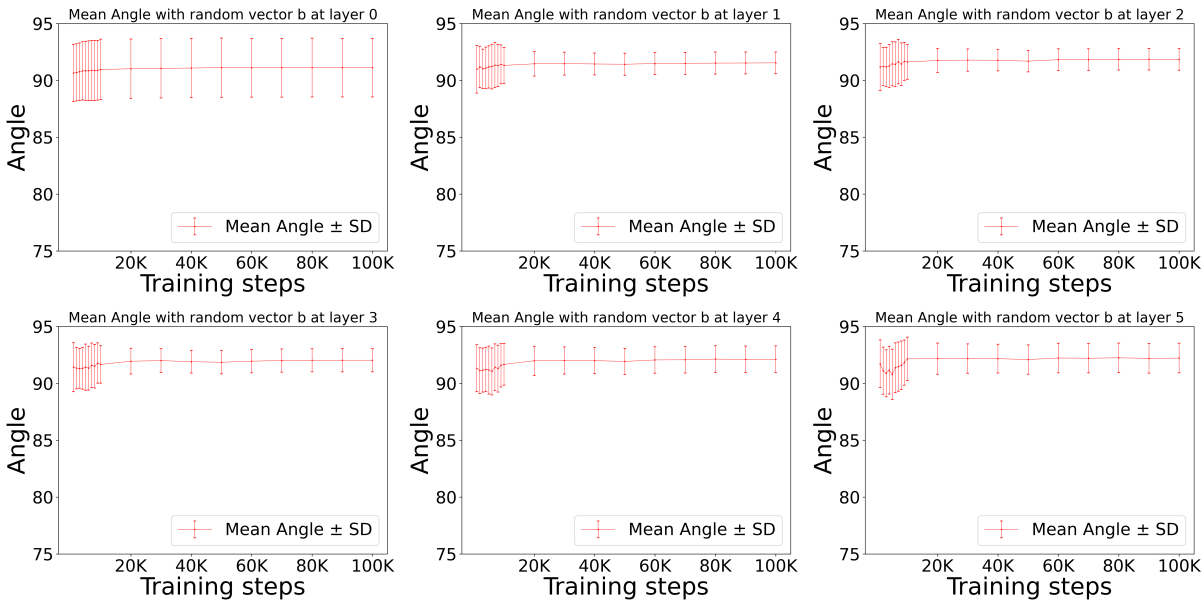


Figure 19: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 70M with RMSNorm during training

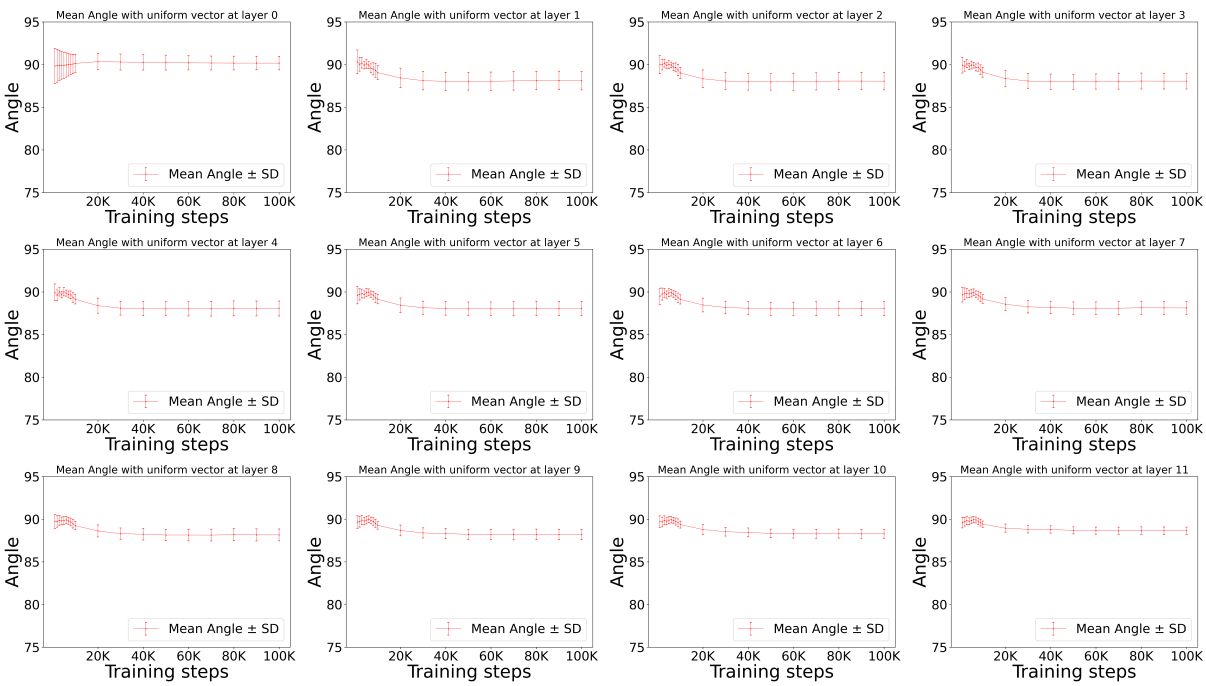


Figure 20: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 160M with LayerNorm during training

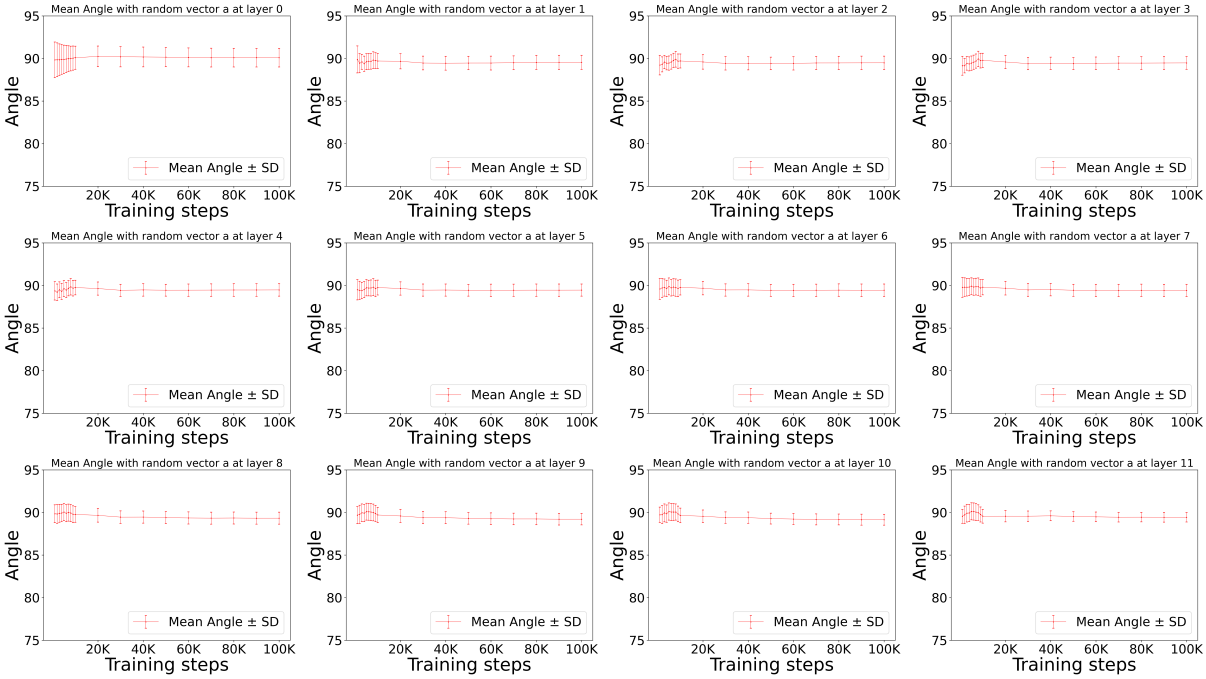


Figure 21: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 160M with LayerNorm during training

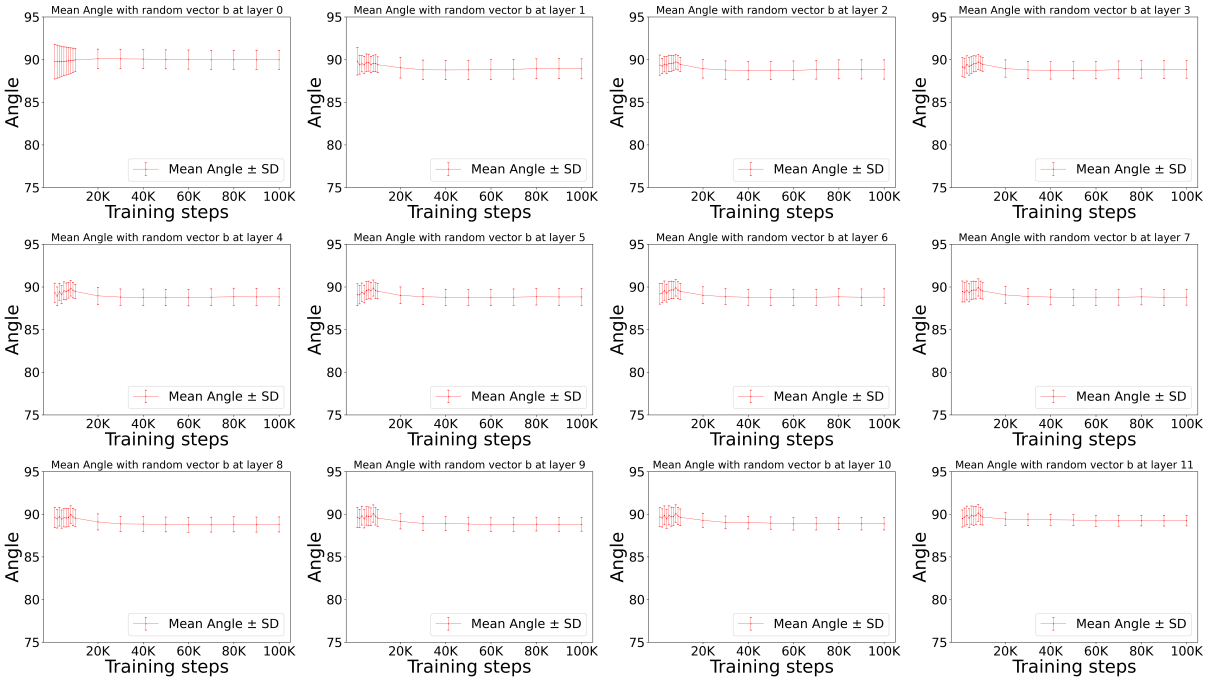


Figure 22: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 160M with LayerNorm during training

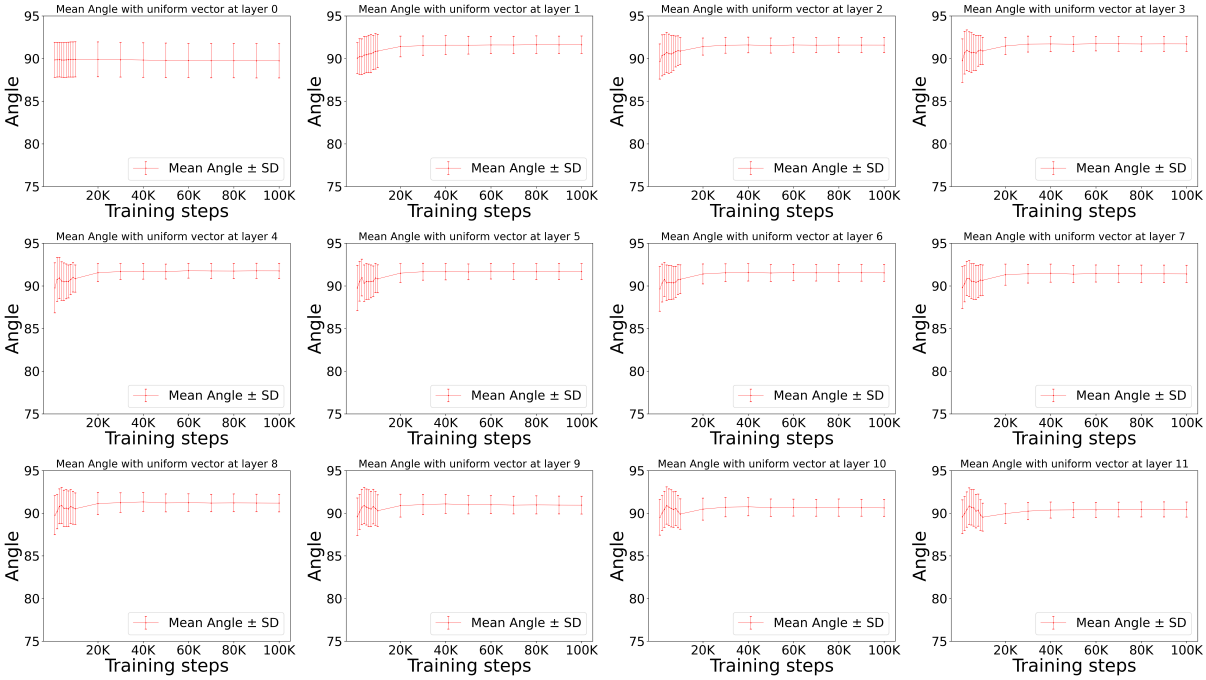


Figure 23: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 160M with RMSNorm during training

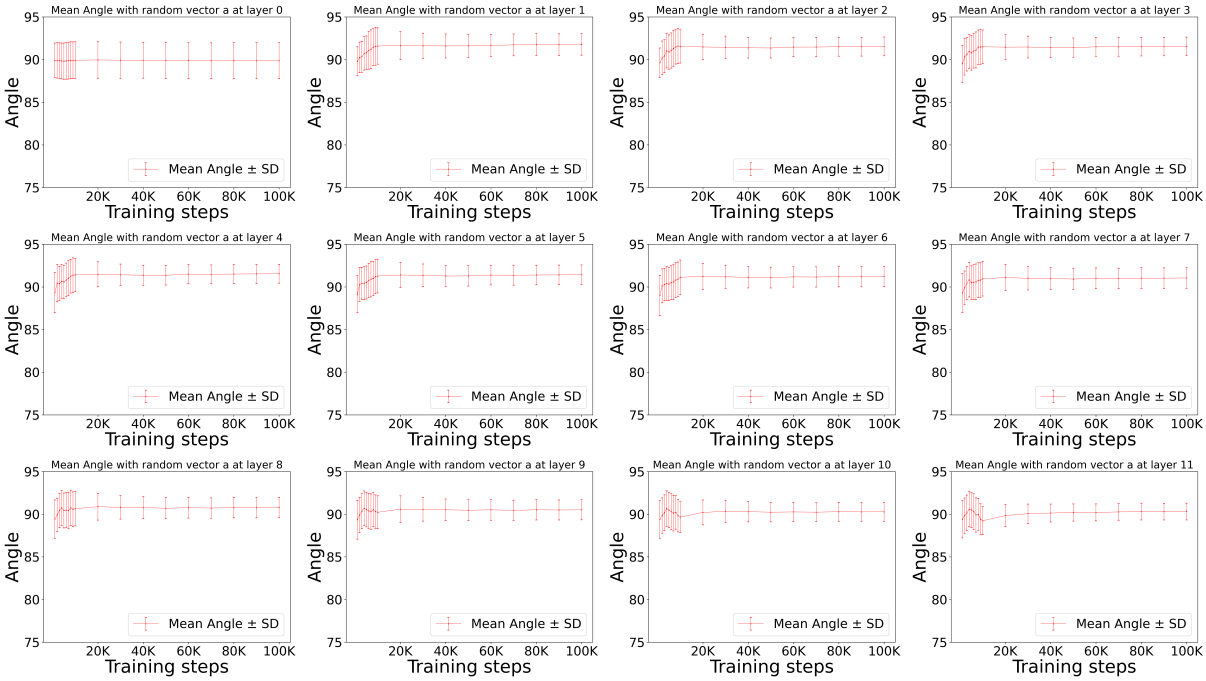


Figure 24: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 160M with RMSNorm during training

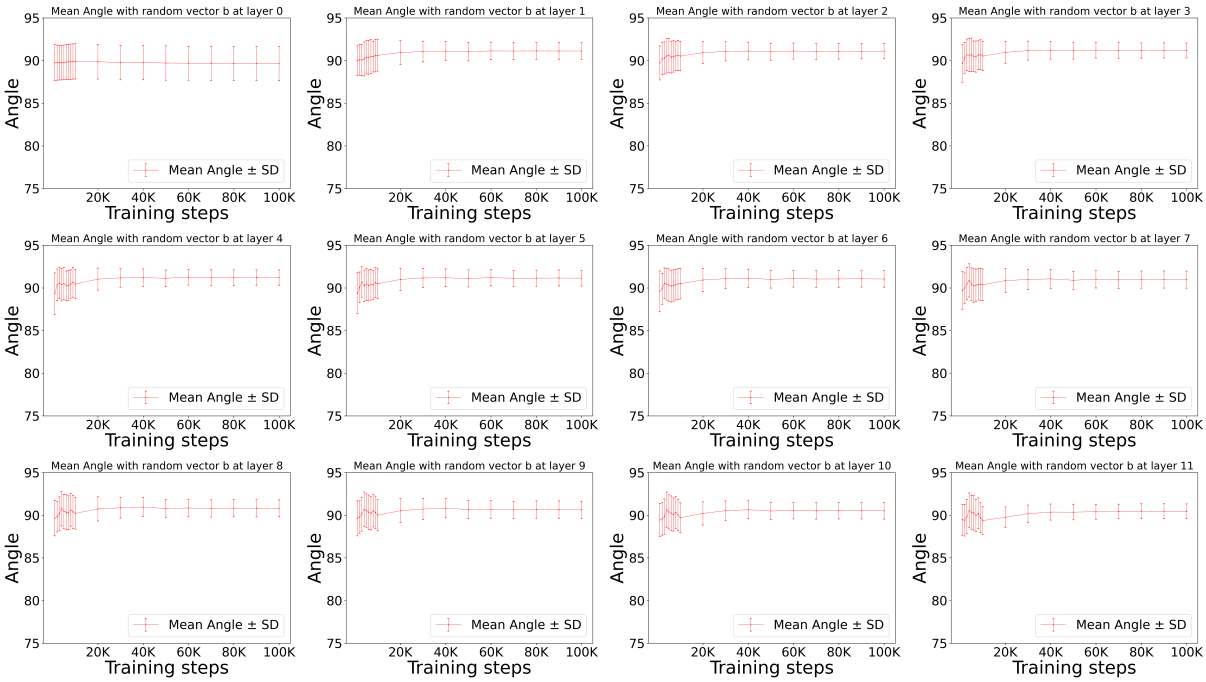


Figure 25: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 160M with RMSNorm during training

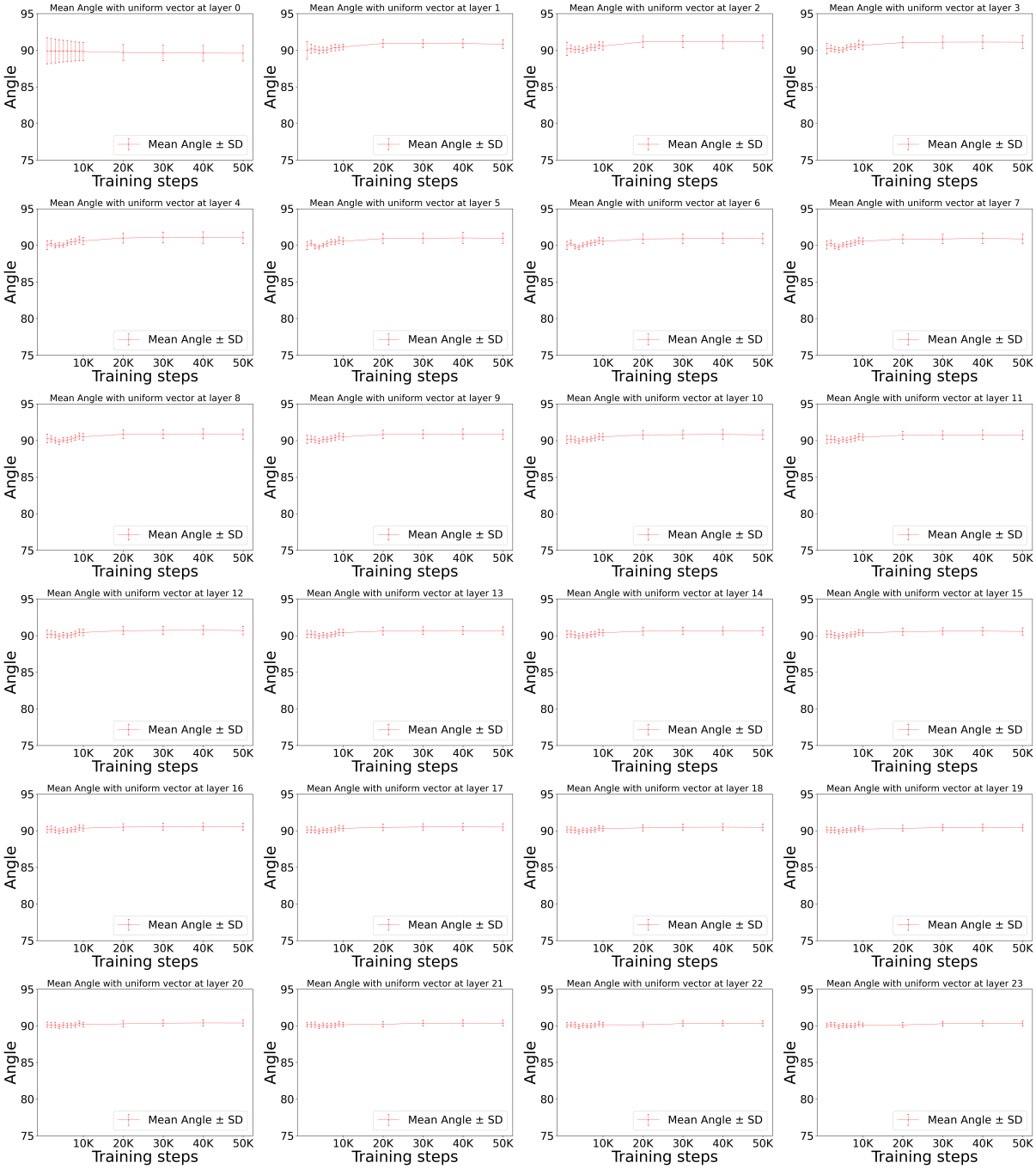


Figure 26: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 410M with LayerNorm during training

Figure 27: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 410M with LayerNorm during training

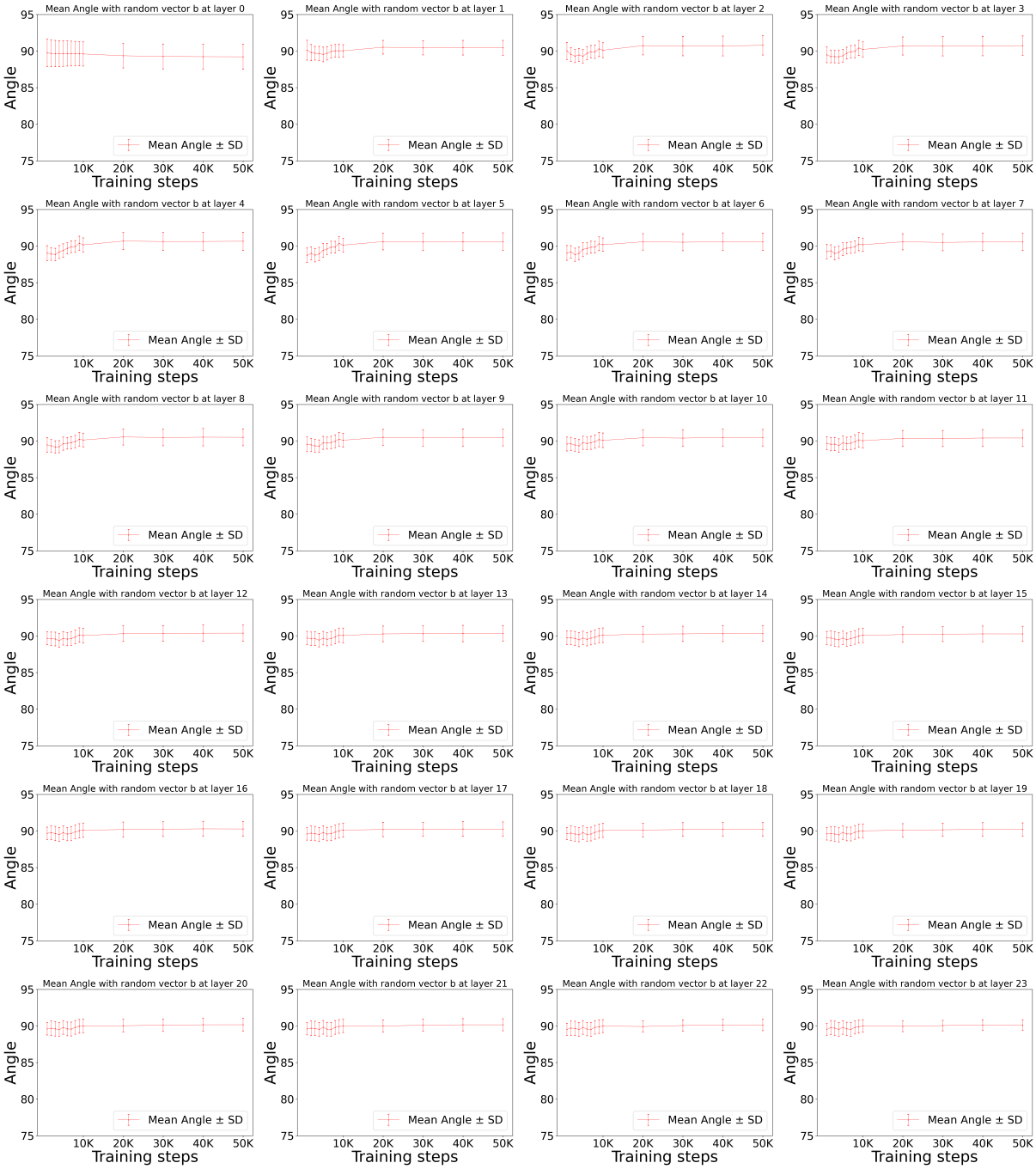


Figure 28: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathrm{b}}$ for all layers in 410M with LayerNorm during training

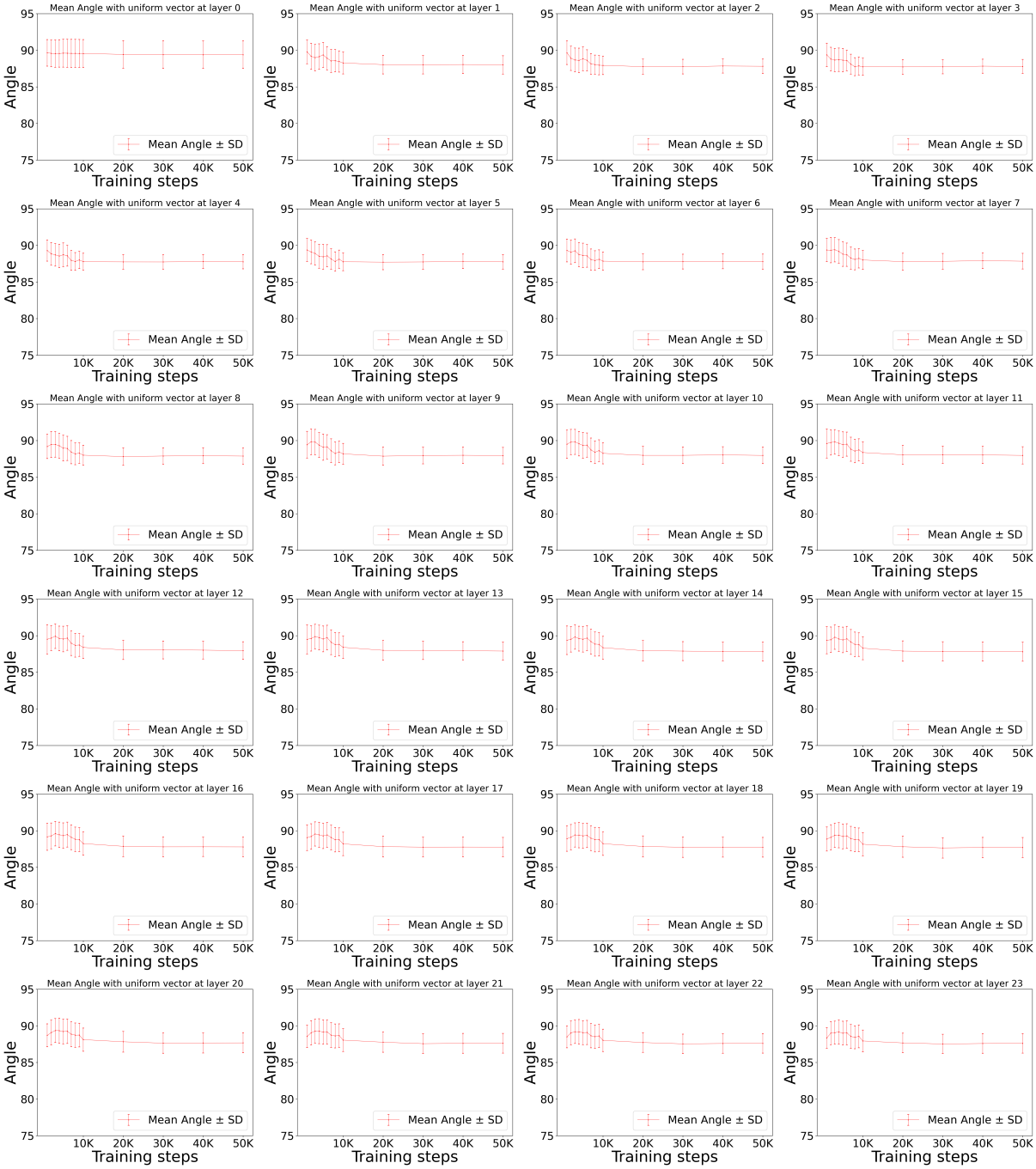


Figure 29: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 410M with RMSNorm during training

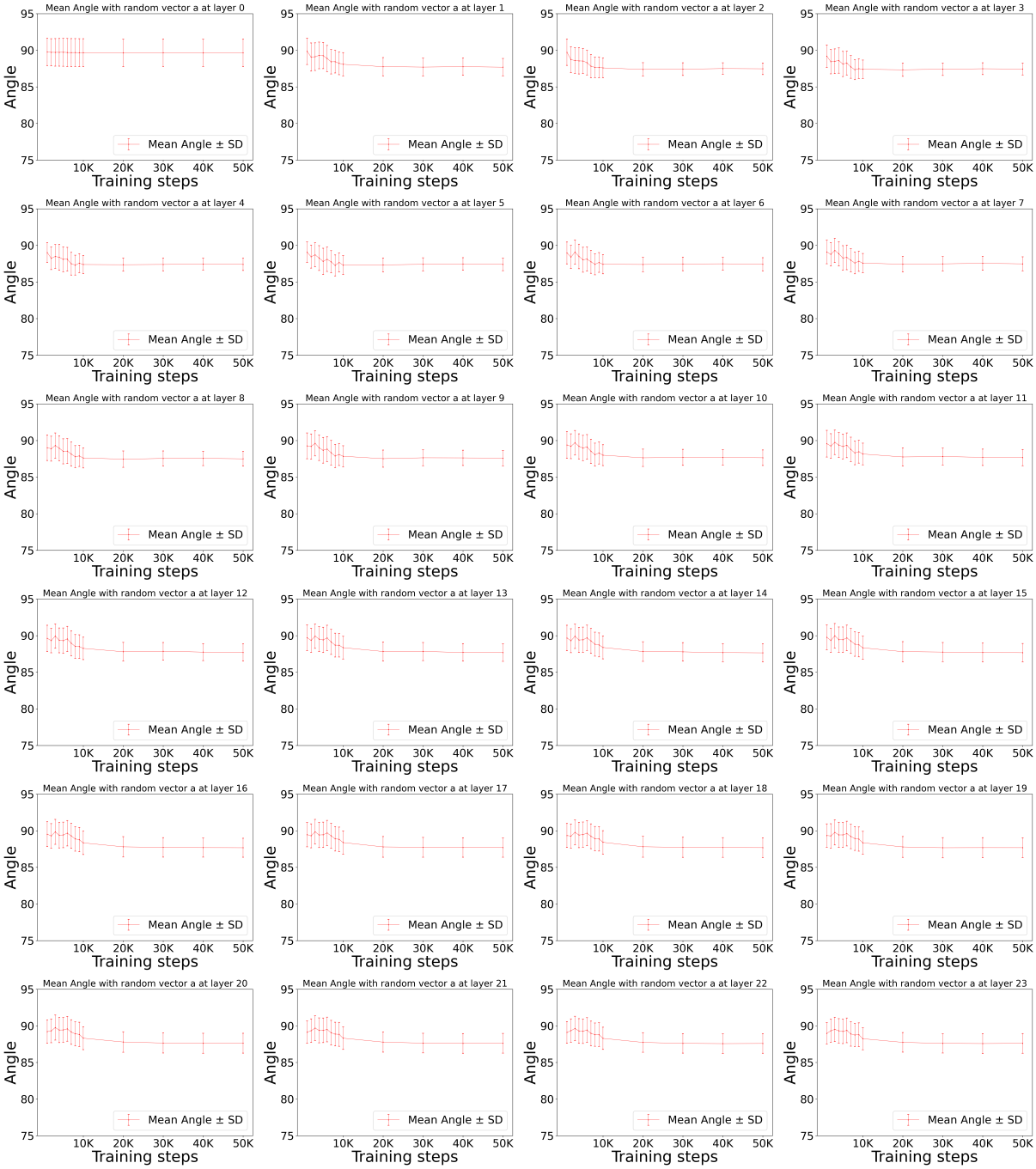


Figure 30: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 410M with RMSNorm during training

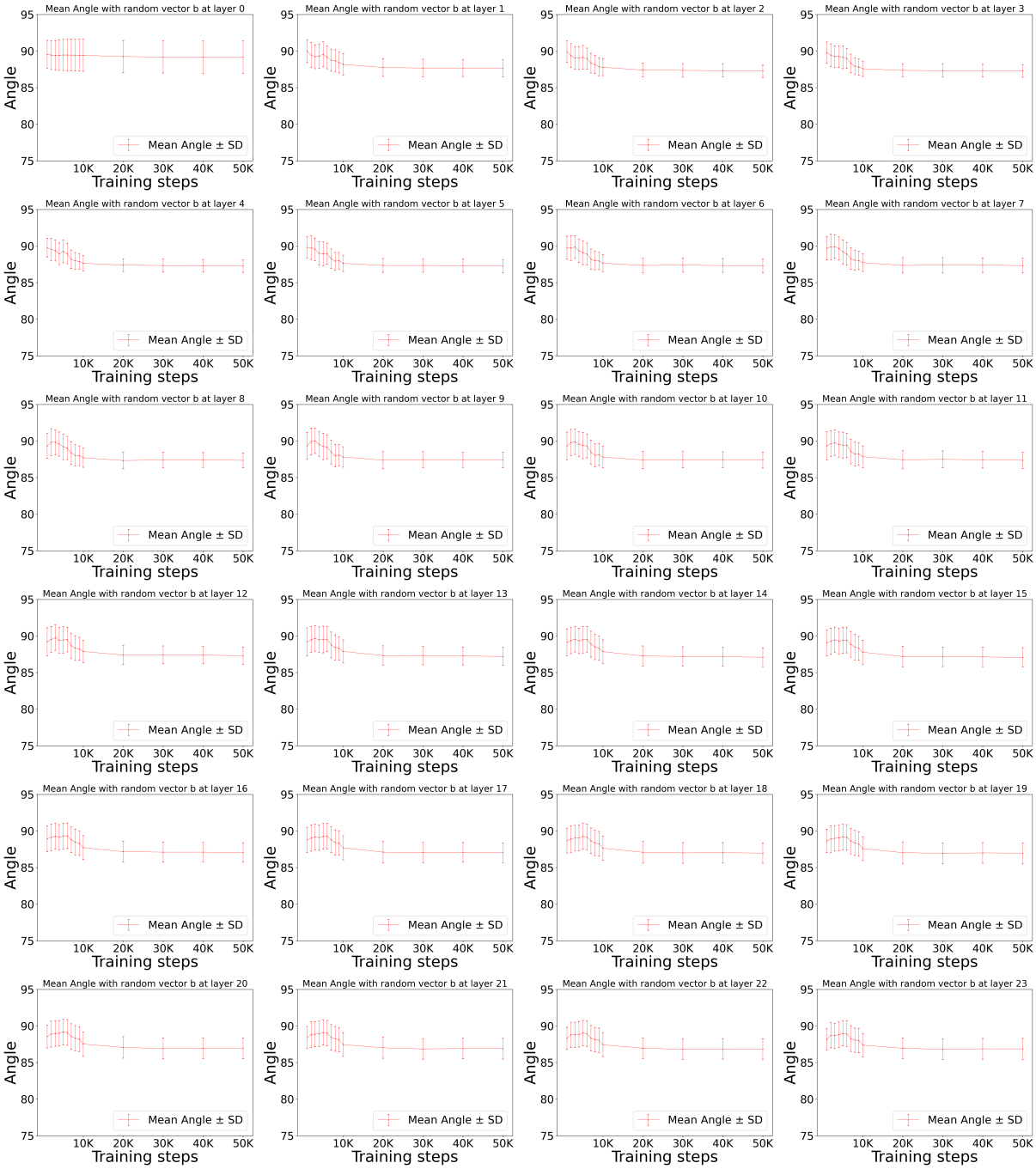


Figure 31: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 410M with RMSNorm during training

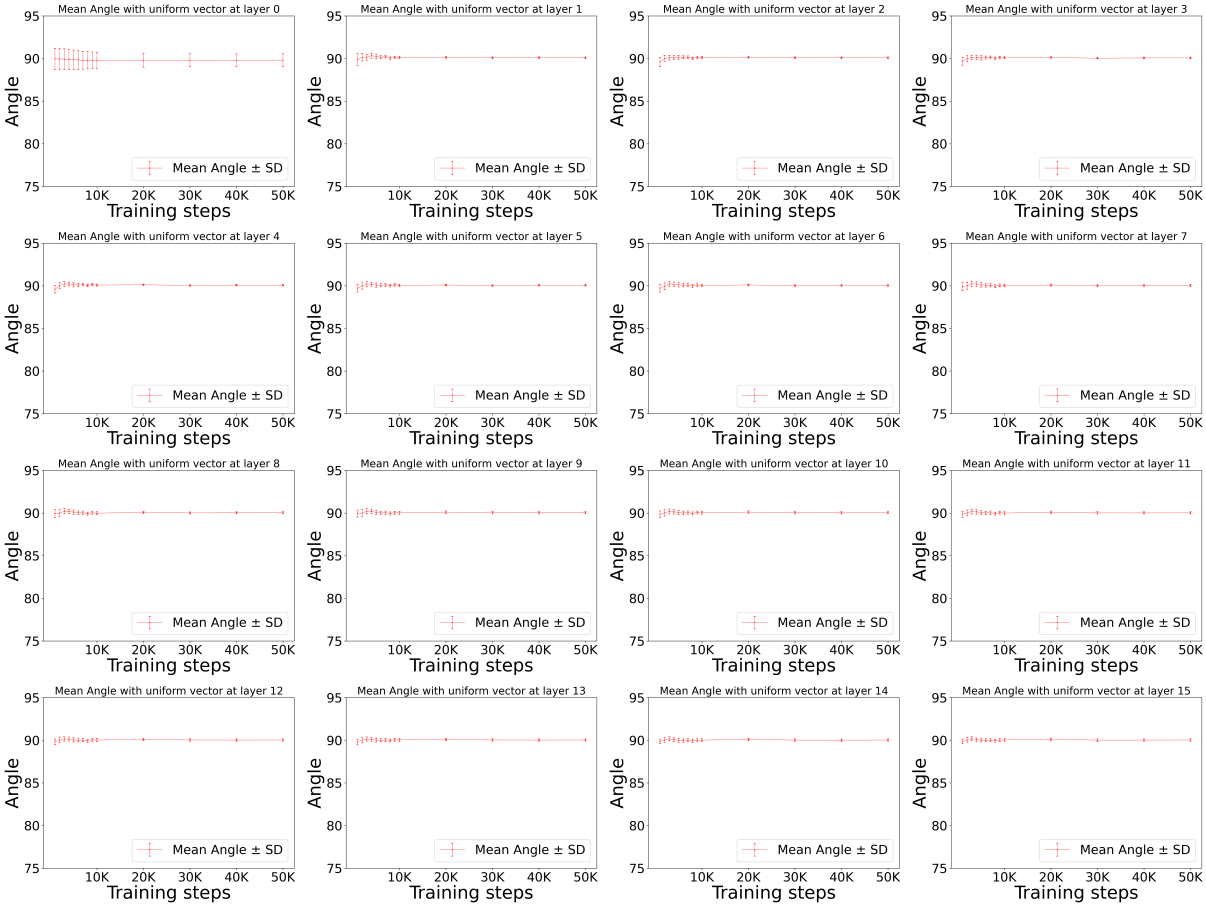


Figure 32: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 1B with LayerNorm during training

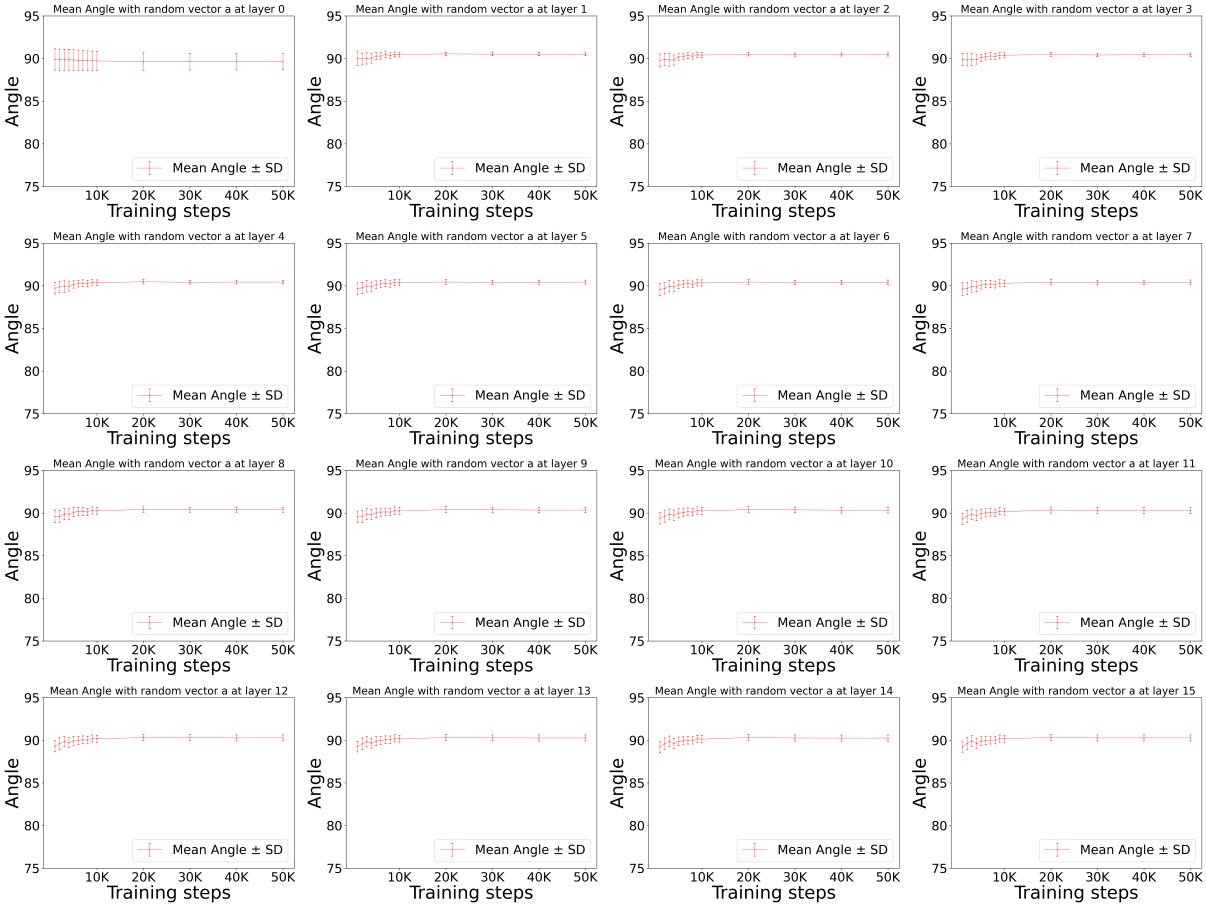


Figure 33: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 1B with LayerNorm during training

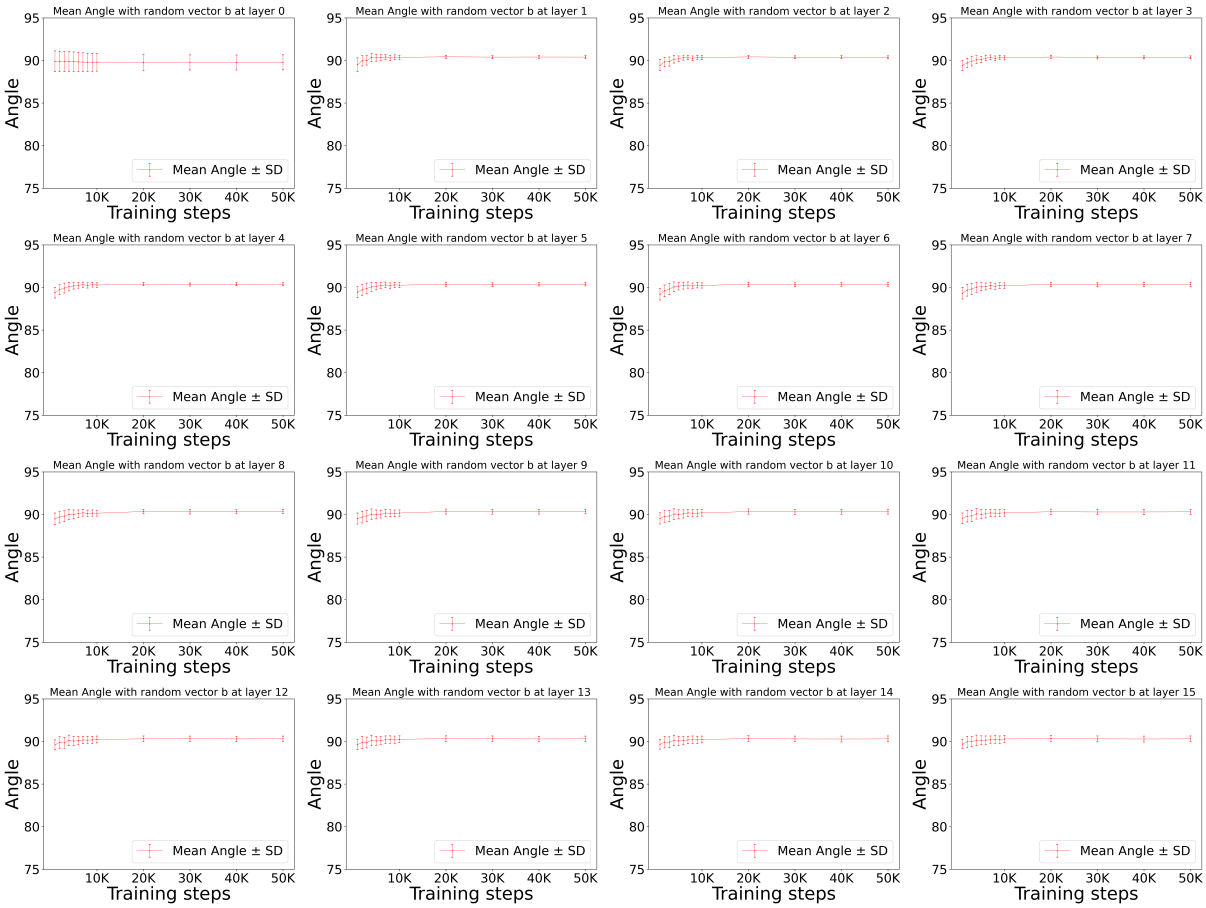


Figure 34: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 1B with LayerNorm during training

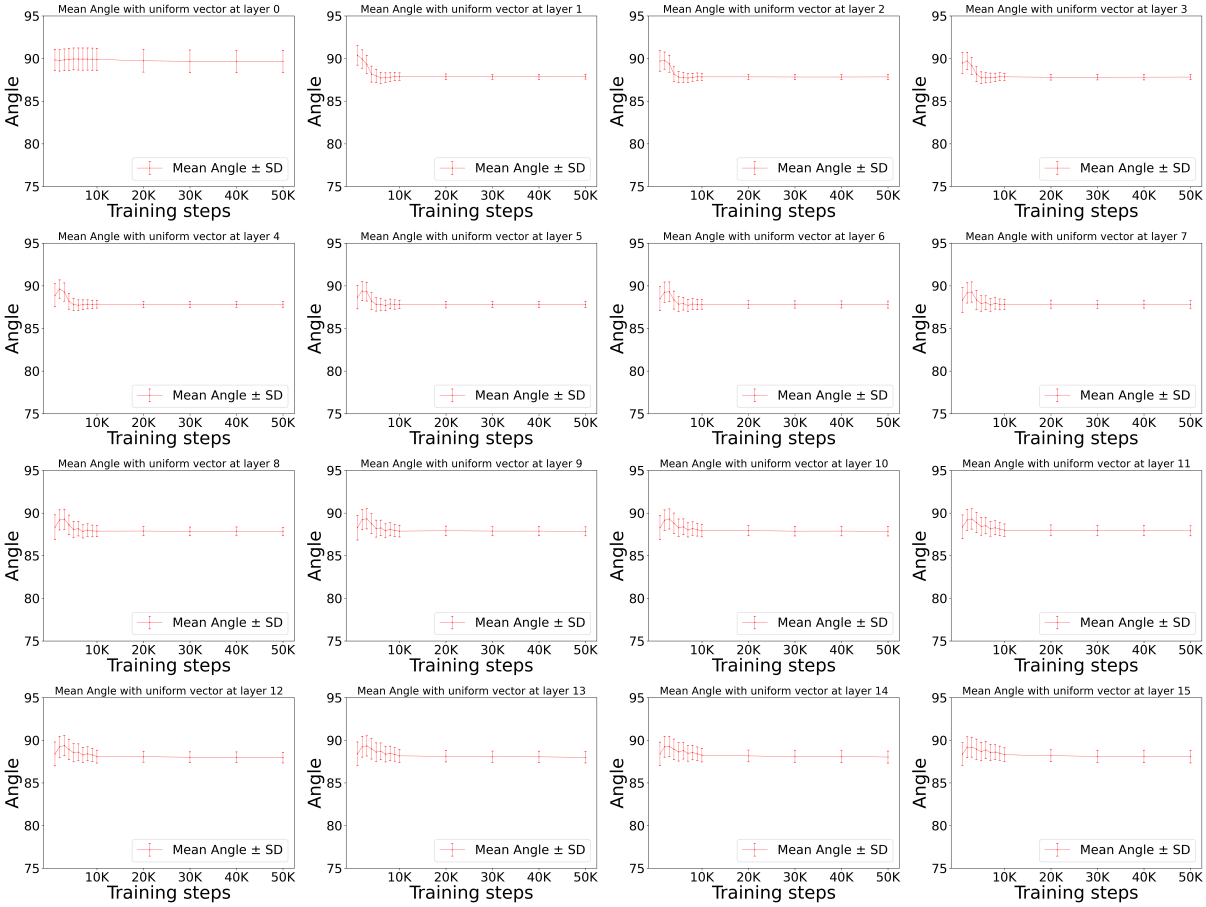


Figure 35: Error bars of angles (in degrees) between Hidden vectors and the *uniform vector* for all layers in 1B with RMSNorm during training

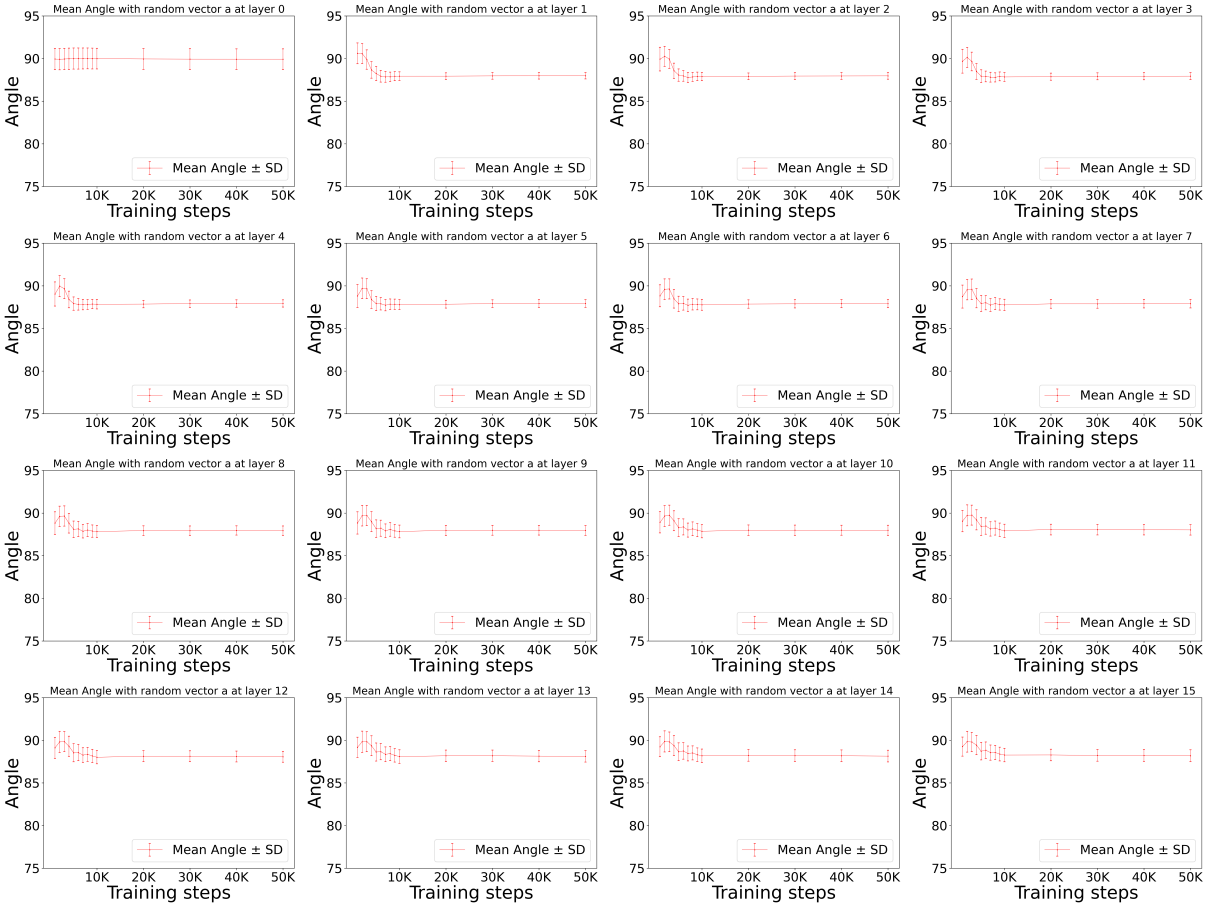


Figure 36: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{a}}$ for all layers in 1B with RMSNorm during training

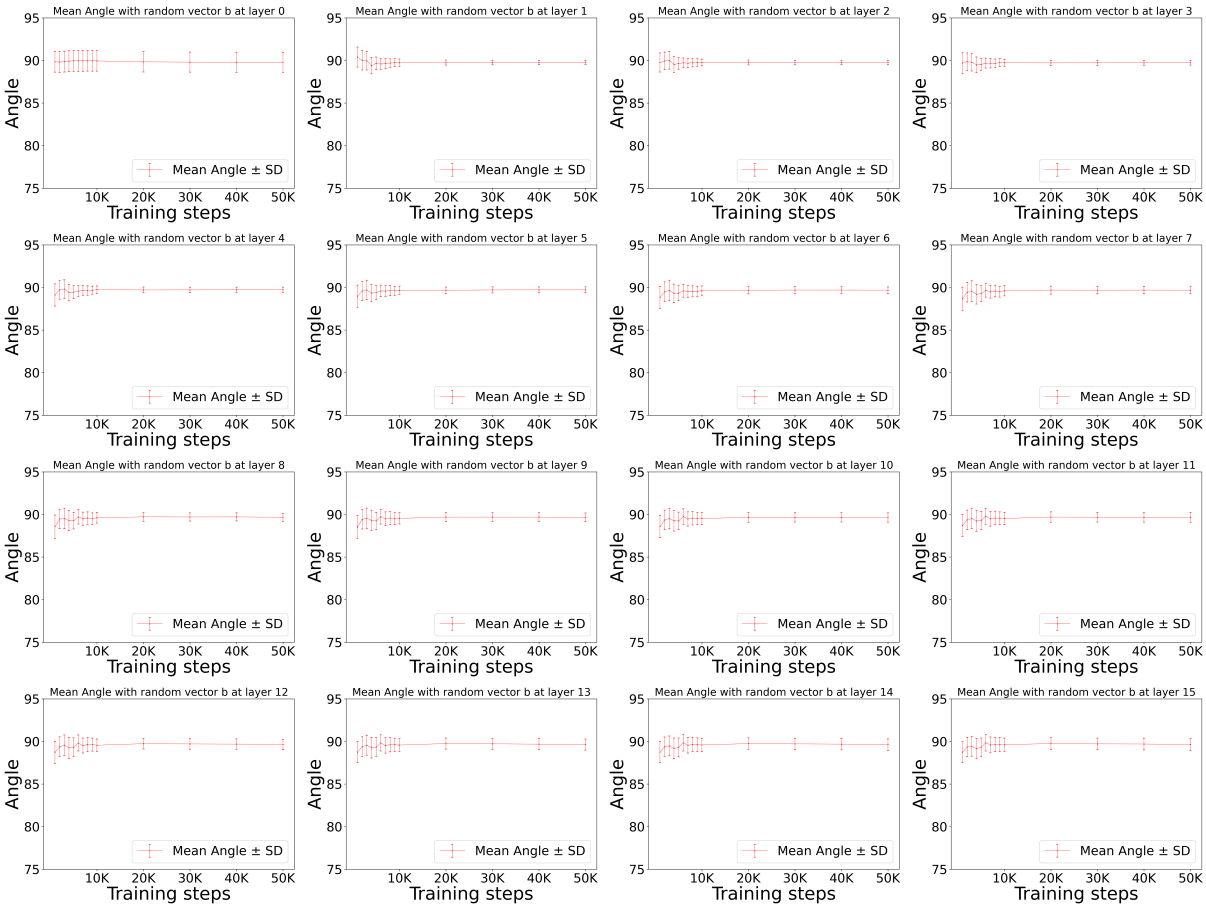


Figure 37: Error bars of angles (in degrees) between Hidden vectors and random vector $\tilde{\mathbf{b}}$ for all layers in 1B with RMSNorm during training