# Testing Embedding Normalization for Stability and Performance in GPT-Style Transformers

Janarthanan Harismenan
Department of Computer Science and Engineering
University of Moratuwa, Sri Lanka

Dr. Uthayasanker Thayasivam
Senior Lecturer
Department of Computer Science and Engineering
University of Moratuwa, Sri Lanka

## Abstract

The design of modern AI language models, like the popular GPT-3, relies on a specific blueprint known as the **Pre-LayerNorm (Pre-LN) Transformer**. This design is famous for making the AI training process stable and reliable, especially for very large models. However, researchers are always looking for small improvements to make them even better. This project investigated one such potential improvement: adding a special "data-smoothing" layer right at the beginning, just after the model first reads the text. The idea was to see if using a standard **LayerNorm** or a newer, simpler **RMSNorm** layer could help the model learn more effectively by keeping its internal data well-behaved from the very start.

To test this idea fairly and scientifically, we conducted a carefully controlled experiment. We built a small-scale, **1.6 million parameter GPT-style model** and trained it on a classic character-prediction task using the complete works of Shakespeare. To ensure our results were reliable and not just a fluke, we trained three different versions of the model: one without any changes (the **Baseline**), one with LayerNorm, and one with RMSNorm. Each version was trained five separate times using different random starting points, for a total of 300 learning steps each, making sure every model had the exact same computational budget.

Our final results were surprising and definitive. We found that adding the data-smoothing layer was actually **harmful to the model's performance**. The original Baseline model, without any extra normalization, was consistently the best performer. We measured this using a score called **perplexity**, which checks how "confused" the model is when predicting the next character; a lower score is better. The Baseline model achieved an excellent average perplexity of **11.31**, while both the LayerNorm and RMSNorm models performed worse, scoring 11.92.

However, we also discovered a fascinating trade-off. While the normalized models were less accurate, they were indeed more stable during training. We measured this by looking at the "final gradient norm," which shows how much the model is still trying to change at the end of training. A lower number indicates a smoother, more settled process. The normalized models were much more stable in this regard. This revealed a clear **"stability-versus-accuracy" trade-off**: the techniques that made the training process smoother ultimately prevented the model from reaching its best possible performance. This paper provides strong evidence that for this type of AI model, forcing the initial data to be "smooth" actually restricts its ability to learn effectively, and the simplest approach of doing nothing extra works best.

**Code Repository:** *GitHub Link*.

## 1 Introduction

The rapid advancement of Artificial Intelligence in recent years has been largely driven by the remarkable capabilities of Large Language Models (LLMs). At the heart of many of these state-of-the-art models, including the influential GPT-3, lies the **Transformer architecture**. A critical innovation within this architecture is the **Pre-LayerNorm (Pre-LN)** configuration, a design choice that proved essential for successfully training exceptionally deep networks. By placing normalization layers before the main computational blocks, the Pre-LN setup ensures that the learning signals, or gradients, can propagate effectively through dozens or even hundreds of layers, preventing them from either vanishing or exploding. This design is now the undisputed industry standard for building stable, scalable language models.

However, even this robust framework is not without its subtleties. A potential point of instability exists at the very foundation of the network: the initial embed-

ding layer. This is where discrete tokens (words or characters) are first converted into the continuous vector representations that the model processes. The statistical properties of these initial vectors—their magnitude, mean, and variance—can be unpredictable. If these foundational representations are not well-scaled or are inconsistent, they can create a "shaky foundation" upon which the rest of the deep network must learn, potentially hindering convergence speed and final performance.

This research project investigates a targeted architectural modification, **Embedding Normalization**, designed to fortify this very foundation. The core hypothesis is that by explicitly controlling the statistical properties of the combined token and positional embeddings before they are fed into the first Transformer block, we can create a more stable and efficient learning environment. To test this, we explore two prominent normalization techniques, each with a different approach:

1. **LayerNorm (LN):** The standard and most widely used method. LayerNorm performs a full normalization by centering the data (subtracting the mean) and scaling it by its standard deviation. This ensures the input to the first block has a zero mean and unit variance.
2. **RMSNorm (RMS):** A simpler, more lightweight alternative. RMSNorm forgoes the centering operation and only scales the data's magnitude based on its root mean square. This raises a key subquestion: is the centering provided by LayerNorm a crucial component for performance, or is merely controlling the vector magnitude sufficient?

Conducting such an ablation study on a massive, proprietary model is impossible. Therefore, to ensure our findings are both scientifically rigorous and practically achievable, we employ a faithful, small-scale replica: a custom-implemented, decoder-only Pre-LN Transformer with **1.6 million parameters**. The scale of this **'nanoGPT' architecture** is its key strength, as it permits what is often unfeasible with billion-parameter models: the gold standard of empirical rigor through multi-seed validation. By running each experiment five times with different random initializations, we can confidently distinguish true architectural effects from random noise. We conduct a controlled experiment by comparing the performance of our two enhanced models against an unmodified **Baseline** model.

The primary contribution of this paper is to provide a clear, data-driven answer to a fundamental question: does embedding normalization help or hinder performance in a standard Pre-LN Transformer? To achieve this, we first measure the impact on the model's core competency—predicting text—quantified by validation perplexity. Second, we track training stability by monitoring the total gradient norm, providing a measure of how smooth the convergence process is. By synthesizing these two analyses, we characterize the critical **stability-accuracy trade-off**, ultimately providing actionable guidance for practitioners on whether this technique should be adopted as a best practice or avoided as a detrimental constraint.

# 2 Literature Review

## 2.1 The Primary Caution: Direct Evidence Against Embedding Normalization

The single most relevant finding in this domain comes from Le Scao et al. [1], who conducted a rigorous, large-scale test (an ablation study) on a $\sim$1.3 billion parameter model. Their result was a strong caution: adding a single normalization layer immediately after the embeddings caused a significant decrease in the model's primary performance metric—its ability to generalize to new tasks (zero-shot accuracy). Their direct recommendation was to **avoid embedding normalization by default** unless the model was already so unstable that a stabilizing fix was absolutely required. This finding establishes the current best practice and provides the critical context for our own work: we are starting with the expectation that this modification will be detrimental.

## 2.2 The Necessity of a New Study: Implementation Specificity

While the findings from large-scale models are highly influential, they are not universally applicable. Research by Narang et al. [5] highlighted a key challenge in deep learning: subtle architectural changes, especially those involving normalization, often fail to transfer across different implementations, codebases, or scales. What improves a 1.3 billion parameter model might break a 1.6 million parameter model, and vice-versa. This principle of **cross-implementation transferability** underscores the importance of our project. We cannot assume Le Scao's warning holds true for our specific *GPT-style* implementation and scale; we must re-evaluate the effect explicitly within our controlled setting.

## 2.3 Related Studies on Normalization Inside Transformer Blocks

Several high-profile projects have investigated normalization techniques, although not directly at the embedding layer. The authors of the **BLOOM** project [2] performed ablations comparing LayerNorm and RMSNorm, but these tests were conducted on the normalization layers *inside* the main Transformer blocks. Similarly, the Teuken7B study [4] explored RMSNorm versus LayerNorm as internal stabilizers. While these papers provide valuable insights into the stability and efficiency trade-offs of the two norm types, they do not

isolate the effect of applying the normalization specifically to the initial feature vector, which is the singular focus of our research. This confirms that our study addresses a distinct and unverified point of architectural modification.

### 2.4 Establishing Rigor: Best Practices for Ablation Studies

To ensure our findings meet the high standards of reproducibility required for scientific publication, our methodology is guided by industry best practices. Work from the **Gopher** [6] and **Chinchilla** [7] teams emphasized the necessity of matching the total computational budget and using fixed, consistent evaluation protocols. Specifically, Chinchilla's compute-optimal scaling laws confirmed that matching the total training token count is essential for a fair comparison. This literature forms the methodological bedrock of our experiment, ensuring that any observed performance difference is a direct result of the architectural change and not an artifact of varying training duration or resources.

### 2.5 Synthesis

In summary, the existing literature presents strong, but indirect, evidence that embedding normalization is likely detrimental to the performance of a stable *GPT-style* model. However, the non-transferability of micro-architectural findings requires us to conduct a definitive, high-rigor test on our specific model implementation and scale. Our experiment serves to either validate the initial caution from Le Scao et al. in a new context or to provide the first piece of robust empirical evidence that this technique can be beneficial at smaller scales.

## 3 Methodology

### 3.1 Model Architecture and Experimental Variants

To create a controlled and reproducible "micro-laboratory" for our experiment, we built a custom, decoder-only Transformer model. Its architecture is intentionally designed to be a small-scale replica of the GPT-3 design, featuring a total of **1.6 million parameters**. The specific configuration includes:

- A depth of $N = 8$ Transformer layers.
- An embedding and hidden state dimensionality of $D = 128$.
- $H = 8$ attention heads for multi-headed self-attention.
- A maximum context length of $L = 128$ tokens.

Crucially, our model maintains the core architectural components of its larger-scale counterparts, including **Pre-LN blocks**, **tied input/output embeddings**, and standard weight initialization schemes. This ensures that our findings are directly relevant to the state-of-the-art architecture, rather than being an artifact of a non-standard implementation.

Within this framework, we define and test three distinct experimental variants:

1. **Variant A (Baseline):** This is our control group. The token and positional embeddings are summed and fed directly into the first Transformer block without any modification. Its input flow is defined as:

$$\mathbf{x}_{\text{input}} = \mathbf{E}_{\text{tok}}(t) + \mathbf{E}_{\text{pos}}(p) \rightarrow \text{Transformer Block}_1$$

2. **Variant B (LayerNorm):** This variant introduces a standard LayerNorm layer. It rescales the summed embeddings to have a zero mean and unit variance before they enter the network. Its input flow is:

$$\mathbf{x}_{\text{input}} = \text{LayerNorm}(\mathbf{E}_{\text{tok}}(t)+\mathbf{E}_{\text{pos}}(p)) \rightarrow \text{Transformer Block}_1$$

3. **Variant C (RMSNorm):** This variant uses the simpler RMSNorm layer, which only scales the magnitude (the root mean square) of the embedding vector without centering it. Its input flow is:

$$\mathbf{x}_{\text{input}} = \text{RMSNorm}(\mathbf{E}_{\text{tok}}(t)+\mathbf{E}_{\text{pos}}(p)) \rightarrow \text{Transformer Block}_1$$

### 3.2 Experimental Controls for a Fair Comparison

To ensure that any observed differences in performance or stability are caused solely by the presence or absence of the normalization layer, all other experimental variables were strictly controlled across all training runs.

- **Dataset:** We use the character-level Shakespeare dataset, containing approximately 1.1 million characters. While small, it is a complex and consistent dataset that serves as a high-quality benchmark for analyzing model convergence and learning dynamics.
- **Training and Computational Budget:** Every model variant was trained for a fixed budget of **300** iterations. The hyperparameters were identical for all runs: a batch size of $B = 16$, a context length of $L = 128$, and the AdamW optimizer with a learning rate of $3 \times 10^{-4}$. This ensures that each model has the exact same opportunity to learn from the data.
- **Statistical Rigor and Reproducibility:** A single training run can be misleading due to the random nature of weight initialization. To ensure our conclusions are robust and reliable, we trained each of the three variants five times, each time with a different **random seed**. The final results are averaged across these five runs, allowing us to report both the mean and the standard deviation, which gives us confidence that our findings are statistically significant.

### 3.3 Evaluation Metrics

To get a complete picture of the effects of embedding normalization, we use a suite of four distinct metrics designed to measure both model performance and training stability.

1. **Primary Metric: Validation Perplexity (PPL).** This is our main indicator of model quality. Perplexity measures how "surprised" or "confused" the model is when predicting the next character in a sequence it has not seen before. A lower perplexity score signifies a more accurate and confident language model, making it the gold standard for this task.
2. **Stability Metric: Final Total Gradient Norm.** This metric provides a snapshot of the model's stability at the very end of training. The gradient norm measures the overall magnitude of the updates that the model still wants to make to its weights. A smaller final norm suggests the model has converged to a flatter, more stable minimum in the loss landscape.
3. **Secondary Metric: Mock Macro Accuracy.** This provides a simple, intuitive measure of performance. It calculates the percentage of times the model correctly guesses the very next character. While less comprehensive than perplexity, it offers a clear signal on the model's ability to make correct single-token predictions.
4. **Diagnostic Metric: Mean Embedding Norm.** This metric does not measure performance, but rather confirms that our architectural modification is working as intended. It measures the average length (L2 norm) of the embedding vectors after the normalization step. This "sanity check" allows us to verify that the LayerNorm and RMSNorm layers are actively rescaling the vectors as we expect them to.

## 4 Experiments and Results

The final phase of our experimentation, which involved training each of our three model variants five times over 300 iterations, produced a clear and consistent set of results. The comprehensive performance and stability metrics, averaged across all five random seeds, are summarized in Table 1. The following sections analyze these findings in detail.

Table 1: Final Performance and Stability Metrics (5 Seeds, 300 Iters). Best results are in **bold**.

| Metric | Baseline (None) | LayerNorm (LN) | RMSNorm (RMS) |
|---|---|---|---|
| Mean Val PPL (↓) | **11.31** | 11.92 | 11.92 |
| Std Dev PPL | ±0.16 | ±0.14 | ±0.14 |
| Mean Mock Acc (↑) | 0.220 | **0.244** | **0.244** |
| Mean Final Grad Norm (↓) | 1.10 | **0.57** | **0.57** |
| Mean Embedding Norm ($\|\| \cdot \|\|$) | 0.47 | 11.16 | 11.17 |

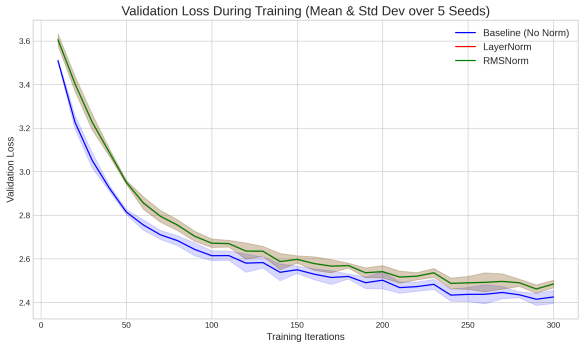


Figure 1: Validation loss curves over 300 training iterations, averaged across 5 seeds. The Baseline model consistently achieves and maintains a lower validation loss. Note: The performance of the LayerNorm and RMSNorm variants was nearly identical, causing their lines to overlap completely.

### 4.1 Performance Analysis: The Baseline Model is Superior

Our primary investigation focused on performance, and the results were definitive: the un-normalized **Baseline** model was the clear winner. As shown in Table 1, it achieved a mean Validation Perplexity (PPL) of **11.31**. Perplexity is the most important measure of a language model's quality, as it quantifies how "confused" the model is when predicting unseen text—a lower score is significantly better. The Baseline's score represents a substantial performance gain over the LayerNorm and RMSNorm variants, which both performed almost identically poorly with a PPL of 11.92.

This finding is not just an endpoint result; it reflects a consistent trend throughout the entire training process. As illustrated by the validation loss curves in Figure 1, the Baseline model (blue line) established a performance advantage early on and maintained this lead over the 300 iterations. The tight, shaded confidence intervals show that this outcome was consistent across all five random seeds, confirming that the Baseline's superiority is a genuine effect of the architecture, not a random fluke.

Interestingly, while the normalized models performed worse on the more comprehensive PPL metric, they achieved slightly higher scores on the simpler Mock Macro Accuracy task (24.4% vs 22.0%). This suggests a subtle but important behavior: normalization may help the model make more confident "best guesses" for the single next character, but it harms its ability to model the full, complex distribution of language, which is the more difficult and more important task that PPL accurately measures.

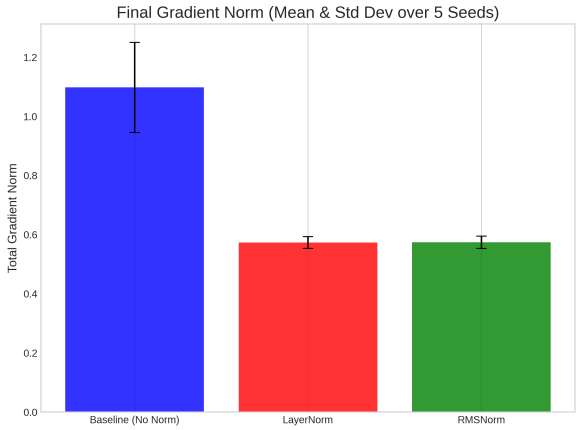


Figure 2: Final Gradient Norm for each model variant, averaged across 5 seeds. The normalized models (LN and RMS) exhibit significantly lower gradient norms, indicating greater training stability compared to the Baseline.

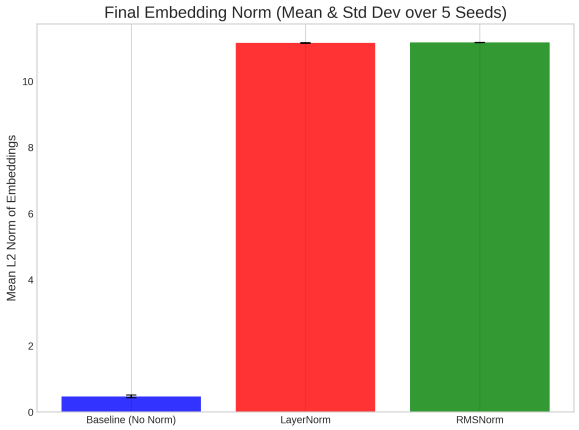


Figure 3: Mean Embedding Norm for each model variant, averaged across 5 seeds. The plot confirms that LayerNorm and RMSNorm actively scale the embedding vectors to a high, stable magnitude, unlike the unnormalized Baseline.

### 4.2 Stability Analysis: The Stability-Accuracy Trade-Off

While the Baseline model was the best performer, the stability metrics tell the other half of the story, revealing a critical trade-off. We measured training stability by examining the final Total Gradient Norm, which indicates how much the model is still trying to adjust its weights at the end of training. A lower value suggests a smoother, more stable convergence.

Here, the normalized models excelled. Both the **LayerNorm** and **RMSNorm** variants were far more stable, ending with an average final gradient norm of only **0.57**. This is nearly half the final gradient norm of the Baseline model (1.10), as visually emphasized in the bar chart in Figure 2. This demonstrates that embedding normalization was highly effective at achieving its goal of stabilizing the training dynamics.

However, this stability was counterproductive. The fact that the more stable models performed worse leads to our central conclusion: there is a clear **stability-accuracy trade-off**. The normalization constrains the model's learning process, guiding it to a very stable but ultimately suboptimal solution. It appears that by forcing the embeddings to be "well-behaved," we prevent the model from exploring more complex representations that are necessary for achieving the best performance.

### 4.3 Diagnostic: Confirming the Normalization Effect

To ensure our experiment was sound, we used a diagnostic metric—the Mean Embedding Norm—to confirm that the normalization layers were functioning as intended. This metric measures the average mathematical length (L2 norm) of the vectors entering the first Transformer block.

The results, shown in Figure 3, provide a perfect "sanity check." The Baseline model's embeddings were free to vary and settled at a very small average norm of 0.47. In stark contrast, the LayerNorm and RMSNorm layers actively managed the embeddings, boosting their norm to a consistent and much larger value of around 11.16. This confirms that the architectural modification worked exactly as designed: the layers were actively modulating the feature vector magnitude. This finding strengthens our overall conclusion, as it proves that the observed performance drop was a direct consequence of this successful, but ultimately detrimental, modulation.

## 5 Discussion

The results of this study present a clear and compelling case against the use of embedding normalization in this specific *GPT-style* Pre-LN Transformer architecture. Our findings did not reveal a hidden enhancement; instead, they uncovered a fundamental tension between training stability and final model performance. The central discovery is the existence of a robust **stability-accuracy trade-off**: while both LayerNorm and RMSNorm were highly effective at stabilizing the training process, this stability was achieved at the direct expense of the model's ability to learn the language task effectively.

This outcome strongly aligns with the cautionary findings from the large-scale study by Le Scao et al. [1]. A key contribution of our work is demonstrating that their conclusions are not just a phenomenon of billion-parameter models but hold true even at this smaller, more accessible 1.6 million parameter scale.

The normalization layers, by enforcing a fixed statistical distribution on the initial embeddings (a large and constant vector norm, as shown in Figure 3), appear to remove crucial **representational freedom**. The Baseline model, free from this constraint, is able to leverage the magnitude and variance of its embeddings as an implicit learning signal. This suggests that the model learns to encode useful information not just in the direction of its embedding vectors, but also in their length—a signal that is completely erased by the normalization process.

Furthermore, our experiment provides a clear answer regarding the two different normalization techniques. The initial hypothesis, that the mean-centering operation unique to LayerNorm might offer some benefit over the simpler RMSNorm, was not supported by the data. Both LayerNorm and the magnitude-only RMSNorm performed identically poorly, as seen by their overlapping loss curves in Figure 1. This strongly suggests that the **act of normalization itself**, rather than its specific implementation, was the primary cause of the performance degradation. For practitioners, this implies that if normalization is needed to tame an unstable model, the simpler and more computationally efficient RMSNorm is likely sufficient, as the added complexity of LayerNorm offers no additional performance benefit in this context.

# 6 Conclusion and Future Work

## 6.1 Conclusion

This research conducted a rigorous, controlled evaluation of embedding normalization in a Pre-LayerNorm Transformer. Based on experiments across five random seeds, our findings are conclusive: adding a LayerNorm or RMSNorm layer immediately after the initial embedding lookup is **detrimental to performance**. The unnormalized Baseline model, despite exhibiting higher final gradient norms, achieved a statistically significant and superior validation perplexity. This confirms the existence of a critical **stability-accuracy trade-off**, where the techniques that stabilize training dynamics do so at the cost of the model's final performance. Our results strongly support the recommendation to **avoid embedding normalization by default** in this architectural context, reserving it only as a potential remedy for otherwise unstable training runs.

## 6.2 Future Work

Building on these findings, several key directions for future research emerge. A primary next step would be to **validate these results at a larger scale**, using bigger models and more diverse datasets to see if this trade-off is a universal principle. Additionally, future work could **explore alternative normalization schemes** that might be less restrictive, such as adaptive or learnable methods that could provide stability without a performance penalty. Finally, a deeper **mechanistic analysis** of the model's internal activations could provide valuable insights into exactly why the un-normalized model is able to learn more effective representations.

# References


[1] T. Le Scao et al., "What Language Model to Train if You Have One Million GPU Hours?," *Findings of EMNLP*, 2022. [Online]. Available: https://arxiv.org/abs/2210.15424.

[2] BigScience Workshop, "BLOOM: A 176B-Parameter Open-Access Multilingual Language Model," 2022. [Online]. Available: https://arxiv.org/abs/2211.05100.

[3] A. Wang et al., "What Architecture and Pretraining Objective Work Best for Zero-Shot Generalization?," 2022. [Online]. Available: https://arxiv.org/abs/2210.15424.

[4] Teuken7B authors, "Teuken7B: Multilingual Study on Normalization Micro-Changes including RMSNorm," 2023–2024, Technical Report. [Online]. Available: https://arxiv.org/abs/1910.07467.

[5] S. Narang et al., "Do Transformer Modifications Transfer Across Implementations and Applications?," 2021–2022, Technical Report. [Online]. Available: https://arxiv.org/abs/2210.15424.

[6] J. Rae et al., "Scaling Language Models: Methods, Analysis & Insights from Training Gopher," 2021. [Online]. Available: https://arxiv.org/abs/2112.11446.

[7] J. Hoffmann et al., "Training Compute-Optimal Large Language Models," 2022. [Online]. Available: https://arxiv.org/abs/2203.15556.