A Comprehensive Study on Optimization Strategies: Initialization, Scheduling, Normalization, and Regularization in Deep Neural Network Training

Pang Liu
jeff.pang.liu@gmail.com

November 16, 2025

1 Introduction

This report presents a comprehensive study on four key dimensions of modern deep network training. The success of deep learning models is critically dependent not just on architecture, but on a synergistic set of optimization choices. This work investigates the interplay between these components to understand their individual and combined effects on model performance across different architectures and tasks.

The four key dimensions investigated are:

- **Weight Initialization** in deep convolutional networks (VGG-19).
- **Learning Rate Scheduling** in residual networks (ResNet-34).
- **Normalization Strategies** in Transformer-based sequence models.
- **Regularization Techniques** (dropout, L2, weight decay, RAdam) in Transformers.

Together, these distinct experimental axes paint a unified picture: **good optimization in deep learning is not controlled by a single "magic" hyperparameter**, but by the *interaction* between initialization, learning rate schedule, normalization choice, and regularization strategy. The subsequent sections present the detailed background, experimental setup, and results from each axis, followed by an integrated conclusion.

2 Background

Before presenting the experimental results, this section provides brief theoretical context for the four optimization pillars under investigation.

2.1 Weight Initialization

The initial values of a network's weights can determine whether it trains successfully or not. Poor initialization can lead to vanishing or exploding gradients, where gradients become too small or too large, respectively, halting the learning process. Methods like **Xavier/Glorot initialization** [1] were designed for saturating activation functions (like sigmoid or tanh), balancing the variance of activations and gradients. However, for non-saturating functions like **ReLU**, **Kaiming/He initialization** [2] was proposed, accounting for the fact that ReLU sets half of its inputs to zero.

2.2 Learning Rate Scheduling

The **learning rate** (LR) is arguably the most important hyperparameter. A learning rate that is too high can cause divergence, while one that is too low can lead to painfully slow convergence or getting stuck in suboptimal local minima. A **learning rate schedule** dynamically adjusts the LR during training. Simple **step decays** (like 'MultiStepLR') or **exponential decays** ('ExponentialLR') lower the LR at fixed intervals. More advanced methods, like **CyclicLR** [6], oscillate the LR between bounds, which can help escape saddle points and find wider, more generalizable minima.

2.3 Normalization in Deep Networks

Normalization layers stabilize training by controlling the distribution of activations.

- **Batch Normalization (BatchNorm)** [8] normalizes activations across the batch dimension. It is highly effective in CNNs but problematic in sequence models due to variable sequence lengths and small batch sizes.
- **Layer Normalization (LayerNorm)** [9] normalizes across the feature dimension, independent of the batch. This makes it the standard for Transformers [7].
- **Position (Pre-LN vs. Post-LN):** In residual architectures like Transformers, the normalization layer can be placed after the residual addition (Post-LN, the original design) or within the residual branch (Pre-LN [10]). Pre-LN is often suggested to provide a cleaner gradient path and more stable training.

2.4 Regularization

Regularization techniques are designed to prevent **overfitting**, a state where the model memorizes the training data but fails to generalize to new, unseen data.

- **Dropout** [11] randomly deactivates a fraction of neurons during training, forcing the network to learn redundant and more robust representations.
- **L2 Regularization** adds a penalty proportional to the squared magnitude of the weights to the loss function, encouraging smaller weights.
- **Weight Decay** is a similar concept, often implemented directly in optimizers like Adam, by subtracting a small fraction of the weight from itself at each step.

Optimizers themselves, like **RAdam** [12] (Rectified Adam), can also have regularizing effects by stabilizing the adaptive learning rate, especially during the initial phase of training.

3 Experimental Setup

The experiments span two different architectures and tasks to provide a broad view of the optimization techniques.

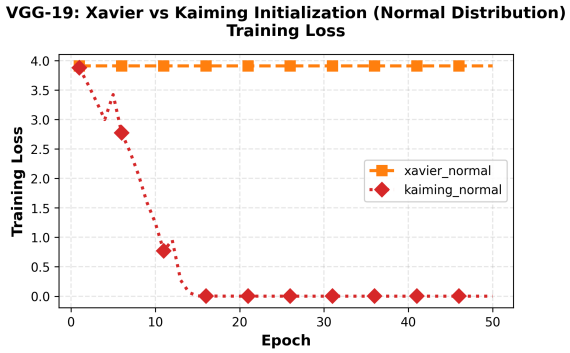
1. **Image Classification:** A VGG-19 [3] and a ResNet-34 [4] are trained on a vision dataset. This setup is used to study how weight initialization affects the ability of a deep ReLU network to learn and how learning rate schedules impact convergence in residual networks.
2. **Machine Translation:** A 6-layer Transformer [7] (4 heads, $d_{\text{model}} = 256$) is used for French-to-English translation on the IWSLT2017 dataset. This setup is used to study normalization and regularization effects on training stability and final BLEU scores.

The main experimental configurations are:

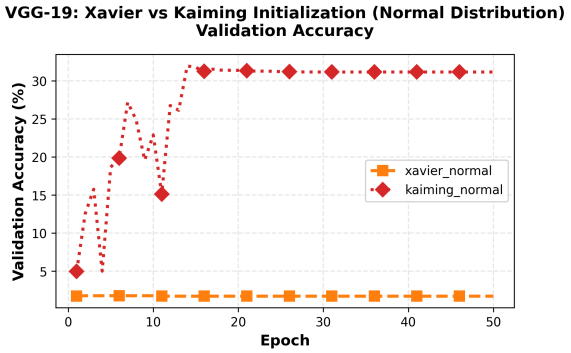
- **CNN / ResNet Experiments**:
  - VGG-19: Initialization experiments using SGD for 50 epochs.
  - ResNet-34: Learning rate schedule experiments using Adam [5] with Kaiming Uniform initialization for 100 epochs.
- **Transformer Experiments**:
  - Transformer: 50 epochs, Adam optimizer (except RAdam experiments), learning rate 0.001, batch size 256.
  - Design axes: dropout vs. no dropout; LayerNorm vs. BatchNorm; pre-norm vs. post-norm; with vs. without LR warmup; additional regularization (L2, weight decay, RAdam).

4 Weight Initialization Comparison (VGG-19)

4.1 Comparison of Initialization Methods (Xavier vs. Kaiming)



(a) Training Loss vs. Epochs (Xavier Normal vs. Kaiming Normal)



(b) Validation Accuracy vs. Epochs (Xavier Normal vs. Kaiming Normal)

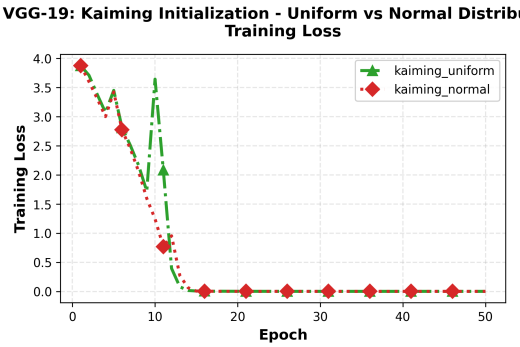
From the plots above, we can observe a **dramatic performance difference** between the two initialization methods.

The **Kaiming Normal initialization** (red diamond line) enables the model to start learning immediately. Its training loss (Figure a) drops rapidly towards zero, and its validation accuracy (Figure b) successfully climbs to over 30%.

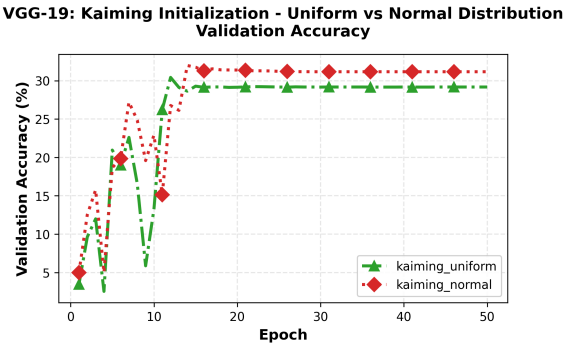
In contrast, the **Xavier Normal initialization** (orange square line) **completely fails to train the network**. As seen in both plots, its training loss is a straight line, and its validation accuracy is also a straight line that has no change.

This result is expected and aligns with established theory [1, 2]. Kaiming initialization is specifically designed for networks using ReLU activation functions, which VGG-19 primarily uses. It prevents vanishing or exploding gradients by scaling the weights correctly. Xavier initialization, designed for sigmoid or tanh activations, is unsuitable for ReLU. This mismatch causes the failure, completely halting the learning process, which is exactly what our experiment shows.

4.2 Comparison of Distribution (Uniform vs. Normal)



(a) Training Loss vs. Epochs (Kaiming Uniform vs. Kaiming Normal)



(b) Validation Accuracy vs. Epochs (Kaiming Uniform vs. Kaiming Normal)

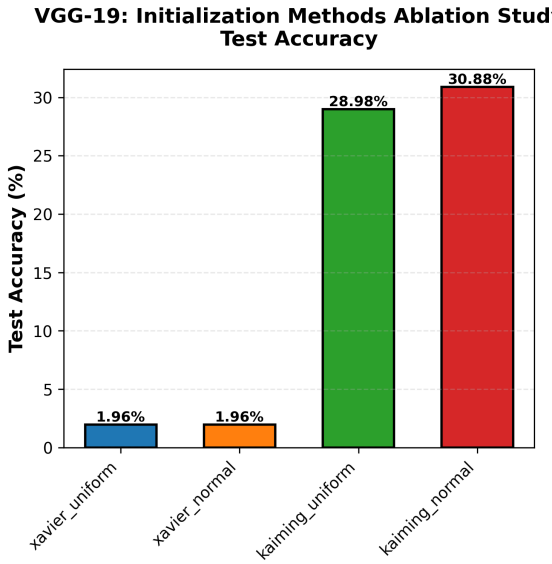
Based on the results from Section 4.1, we furthermore selected the Kaiming initialization method to compare its Uniform and Normal distribution variants.

From the training plots, we can see that while **both methods ultimately enable the network to train, their training dynamics are quite different.** The **Kaiming Normal** (red diamond line) demonstrates a relatively smooth and stable convergence path.

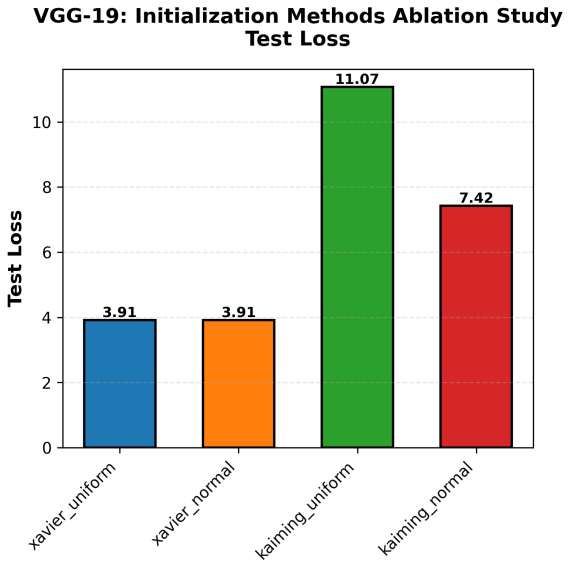
In contrast, the **Kaiming Uniform** (green triangle line) exhibits **significant instability** during the initial training phase (around epochs 5–10). We can clearly observe a **large punctuation in its training loss** (Figure a) and a corresponding **sharp drop and recovery in its validation accuracy** (Figure b).

Although both methods eventually converge, this initial instability of Kaiming Uniform suggests it may struggle to find an optimal solution as effectively as Kaiming Normal, and its unstability might cause more unknown risks, therefore in this experiment Kaiming Normal performs better and is suggested to use firstly.

4.3 Ablation Study: Final Test Performance



(a) Final Test Accuracy (Initialization Methods on VGG-19)



(b) Final Test Loss (Initialization Methods on VGG-19)

Table 1: VGG-19 Initialization Method Performance Summary

Method	Test Accuracy (%)	Test Loss
Kaiming Normal	30.88	7.4210
Kaiming Uniform	28.98	11.0686
Xavier Uniform	1.96	3.9130
Xavier Normal	1.96	3.9130

The bar charts and Table 1 summarize the final test performance of all four methods.

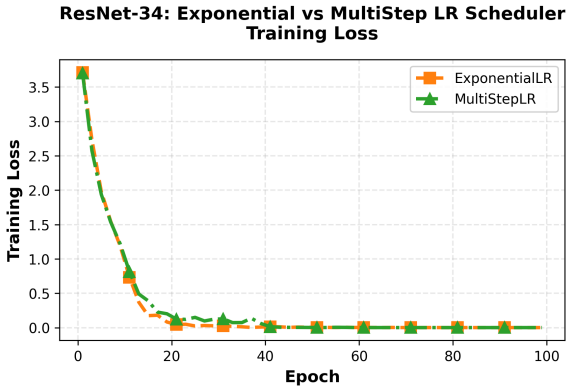
The results confirm the conclusions we drew from our training plots. Kaiming Normal is the definitive winner, achieving the highest test accuracy of 30.88%. Kaiming Uniform follows at 28.98%.

Furthermore, it is critical to observe the Test Loss (Figure b). While Kaiming Normal's accuracy is only ∼2% higher than Kaiming Uniform's, its final test loss is significantly lower (7.42 vs 11.07). This suggests that the model trained with Kaiming Normal is not only more accurate but also more confident in its predictions, indicating superior generalization. This strong result correlates with the more stable training dynamic we observed in Section 4.2.

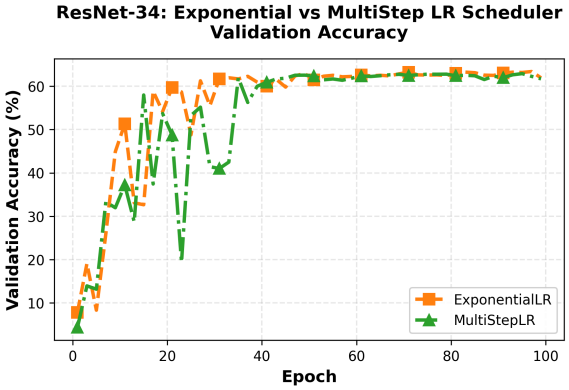
As observed during training, both Xavier methods are complete failures, achieving only 1.96% accuracy. This is equivalent to random guessing on this dataset. This experiment clearly demonstrates that proper initialization is not just a minor optimization but a critical requirement for training deep networks like VGG-19. A wrong choice (e.g., using Xavier for ReLU) will result in the model failing to learn at all.

5 Learning Rate Scheduling Comparison (ResNet-34)

5.1 Fixed Schedule Comparison (Exponential vs. Multistep)



(a) Training Loss vs. Epochs (Exponential vs. Multistep)



(b) Validation Accuracy vs. Epochs (Exponential vs. Multistep)

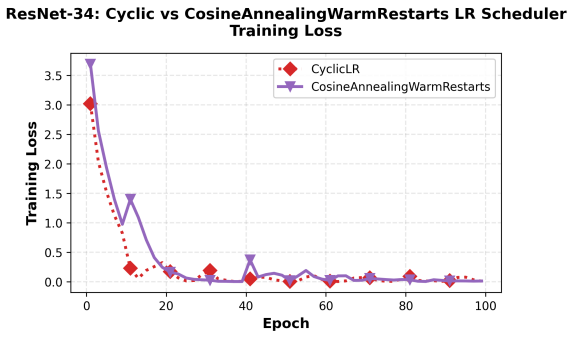
This comparison evaluates two "fixed-decay" schedulers: **ExponentialLR** (orange square line) and **MultiStepLR** (green triangle line).

From the Training Loss plot (Figure a), the two methods perform almost identically, with their curves highly overlapping.

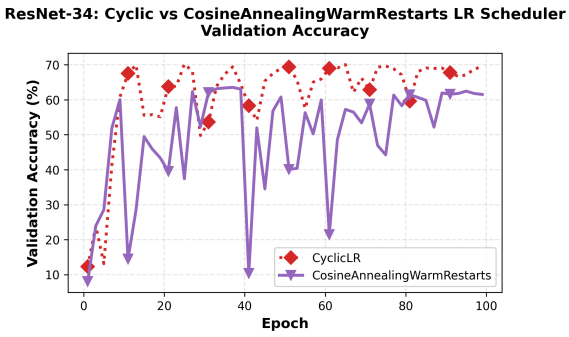
The Validation Accuracy plot (Figure b) reveals their different training dynamics. The ExponentialLR curve is relatively smooth. In contrast, MultiStepLR exhibits **significantly larger and more frequent fluctuations**.

Although ExponentialLR appears more stable during validation, the final test accuracy from our ablation study (see Table 2) reveals that **MultiStepLR is the winner of this comparison**, achieving **61.62%** accuracy, slightly outperforming ExponentialLR's **60.98%**. This indicates the validation-time volatility did not negatively impact its final generalization.

5.2 Cyclic Schedule Comparison (Cyclic vs. Cosine Annealing)



(a) Training Loss vs. Epochs (Cyclic vs. Cosine Annealing)



(b) Validation Accuracy vs. Epochs (Cyclic vs. Cosine Annealing)

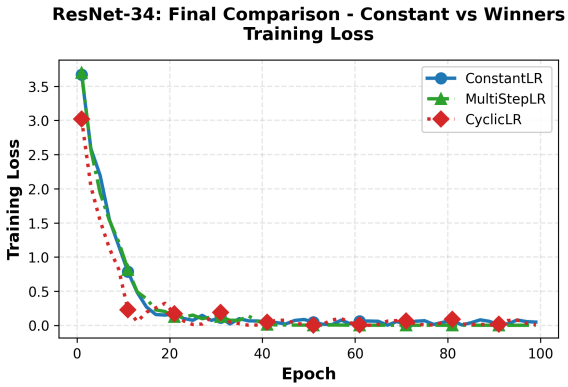
This section compares two schedulers that use cycles and restarts: **CyclicLR** [6] (red diamond line) and **CosineAnnealingWarmRestarts** (purple inverted triangle line).

The validation accuracy plot shows a **striking performance difference**. The CyclicLR curve climbs steadily and maintains a high accuracy, demonstrating a stable path to a good minimum.

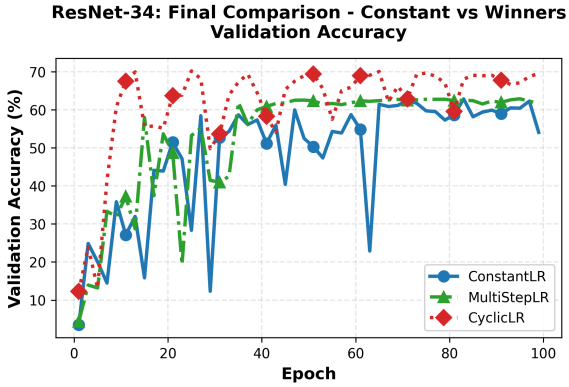
In contrast, CosineAnnealingWarmRestarts shows **extremely punctuate**. This is a direct consequence of its design, the Warm Restarts abruptly reset the learning rate to its maximum value. This large LR spike "kicks" the optimizer out of its current position, causing the validation accuracy to plummet. While the model recovers as the LR anneals, these repeated, violent shocks prevent it from settling into a high-quality minimum.

The final test results (Table 2) confirm this. **CyclicLR is the clear winner of this comparison**, achieving **69.45%** accuracy, which is decisively better than the volatile CosineAnnealingWarmRestarts (62.09%).

5.3 Final Comparison (Constant vs. Winner 1 vs. Winner 2)



(a) Training Loss vs. Epochs (Constant LR vs. Winner 1 vs. Winner 2)



(b) Validation Accuracy vs. Epochs (Constant LR vs. Winner 1 vs. Winner 2)

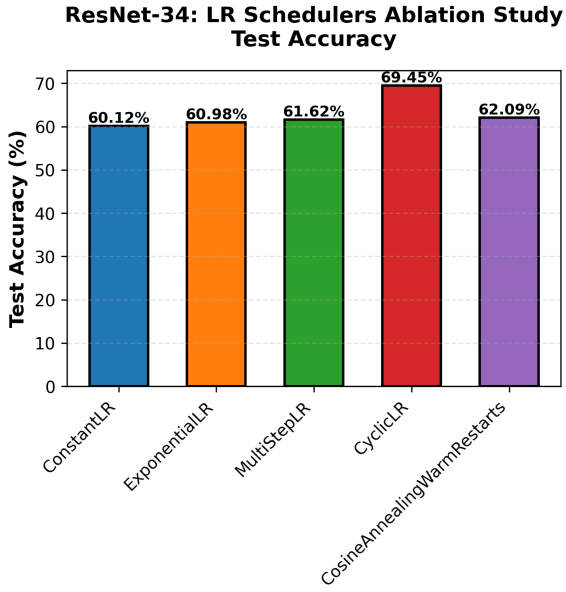
This final comparison pits the baseline (**ConstantLR**, blue circle line), the best fixed scheduler from 5.1 (**MultiStepLR**, green triagle line), and the best cyclic scheduler from 5.2 (**CyclicLR**, red diamond line) against each other.

(Note: MultiStepLR is chosen as 'Winner 1' because its final **test accuracy** of 61.62% was higher than ExponentialLR's 60.98%, as shown in the final ablation study (Section 5.4), despite its volatile validation curve.)

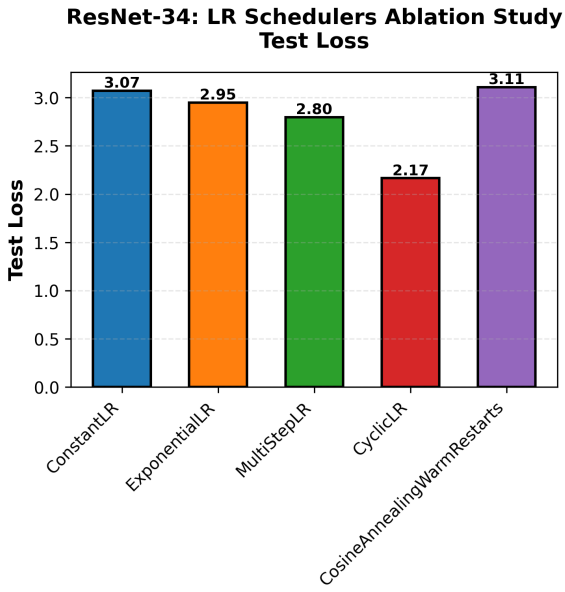
Although the training loss (Figure a) is not significantly different between the schedulers in the final epochs, the validation accuracy (Figure b) tells a clearer story. Both schedulers, MultiStepLR and CyclicLR, clearly outperform the baseline ConstantLR. In fact, the ConstantLR curve is not only the lowest in terms of accuracy but also shows obvious peaks and valleys, indicating an unstable training path.

Comparing the two winners, CyclicLR shows a much better overall performance. Its accuracy curve climbs far higher than MultiStepLR, which appears to hit a performance plateau after epoch 60. This demonstrates CyclicLR's superior ability to find a better, more generalizable minimum.

5.4 Ablation Study: Final Test Performance



(a) Final Test Accuracy (Learning Rate Schedulers on ResNet-34)



(b) Final Test Loss (Learning Rate Schedulers on ResNet-34)

Table 2: ResNet-34 LR Scheduler Performance Summary

Scheduler	Test Accuracy (%)	Test Loss	vs Baseline
CyclicLR	69.45	2.1654	+9.33%
CosineAnnealingWarmRestarts	62.09	3.1062	+1.97%
MultiStepLR	61.62	2.7977	+1.50%
ExponentialLR	60.98	2.9458	+0.86%
ConstantLR (Baseline)	60.12	3.0683	+0.00%

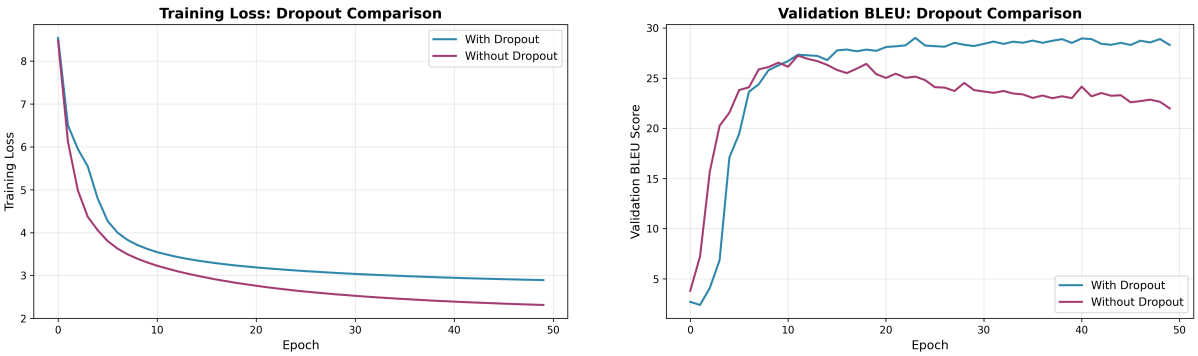
The final table shows the performance of five different learning rate (LR) schedulers on a ResNet-34 model. The ConstantLR scheduler was used as the baseline, or starting point, and it achieved a test accuracy of 60.12%. The results clearly show that the CyclicLR scheduler performed the best by a large margin. It reached a much higher test accuracy of 69.45% and had the lowest test loss. This was a significant 9.33% improvement over the baseline. The other schedulers, like CosineAnnealingWarmRestarts and MultiStepLR, also performed better than the baseline, but their improvements were much smaller (only about 1–2%). In summary, this test indicates that CyclicLR was by far the most effective scheduling method.

6 Normalization and Regularization in Transformer Training

In this section, we shift from convolutional and residual networks to Transformer-based sequence models. The following experiments investigate how dropout, normalization type and position, learning rate warmup, and additional regularization affect training stability and translation quality.

6.1 Effect of Dropout Regularization

6.1.1 Training Dynamics Comparison

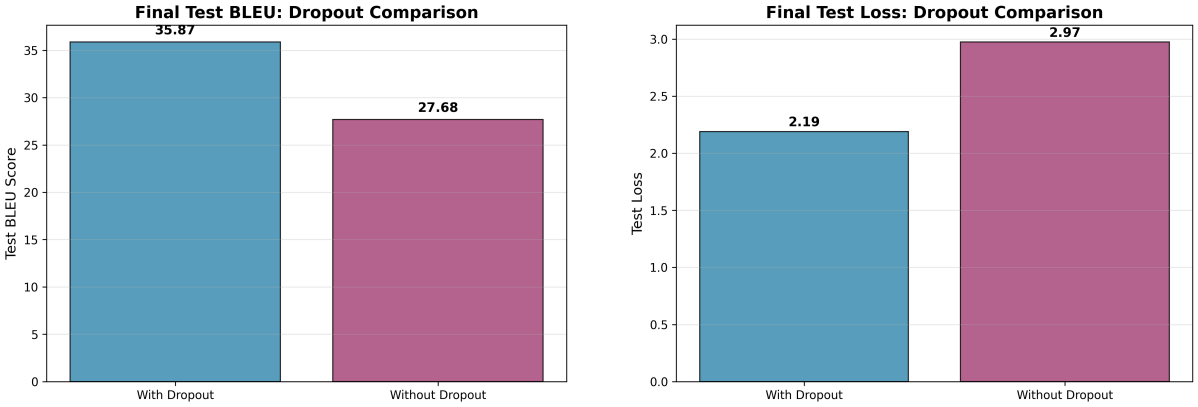


(a) Training Loss vs. Epochs

(b) Validation BLEU vs. Epochs

Figure 8: Dropout Effect: Training Loss and Validation BLEU Score Comparison

6.1.2 Final Test Performance



(a) Final Test BLEU Score

(b) Final Test Loss

Figure 9: Dropout Effect: Final Test Performance Comparison

6.1.3 Analysis and Discussion

The experimental results demonstrate a **significant performance difference** between models trained with and without dropout regularization [11].

The model **with dropout** achieved a test BLEU score of **35.87**, while the model **without dropout** only achieved **27.68**. This represents an **improvement of approximately 8 BLEU points**, which is substantial in machine translation tasks.

Examining the training dynamics (Figure 8), we observe a critical pattern: the model without dropout shows **clear signs of overfitting**. Its training loss continues to decrease throughout

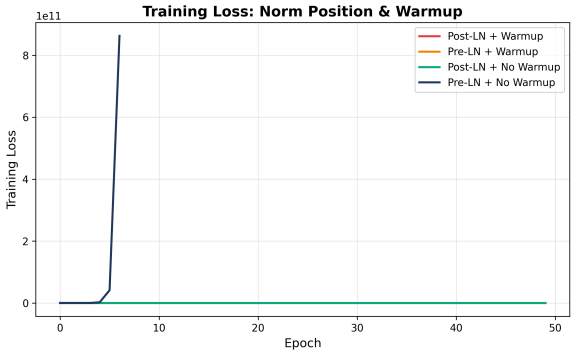
training (reaching lower values than the dropout model), and its training accuracy climbs as high as 83%. However, this does not translate to better generalization. The validation BLEU score plateaus around epoch 12 and even starts to decline slightly, indicating the model is memorizing the training data rather than learning generalizable translation patterns.

In contrast, the model with dropout maintains a healthier balance between training and validation performance. Although its training loss is slightly higher, this controlled underfitting leads to much better generalization on unseen test data.

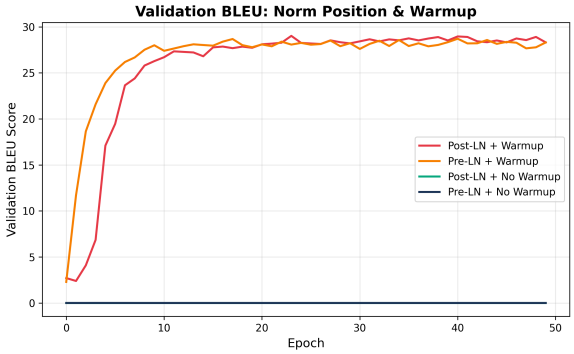
Dropout randomly deactivates neurons during training, preventing units from co-adapting too much to specific training examples. This forces the network to learn more robust features that work even when some neurons are missing. The result is a model that generalizes better to new translations, as evidenced by the 8-point BLEU improvement.

6.2 Comparison of Normalization Methods

6.2.1 Effect of Normalization Position on Learning Rate Warmup

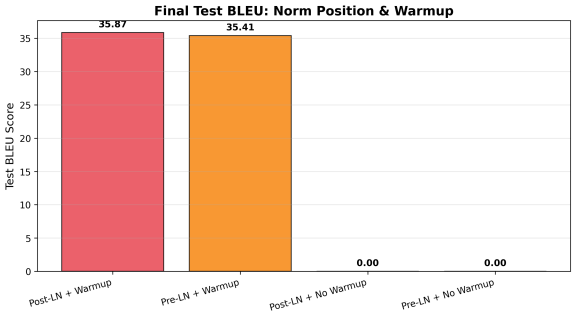


(a) Training Loss vs. Epochs

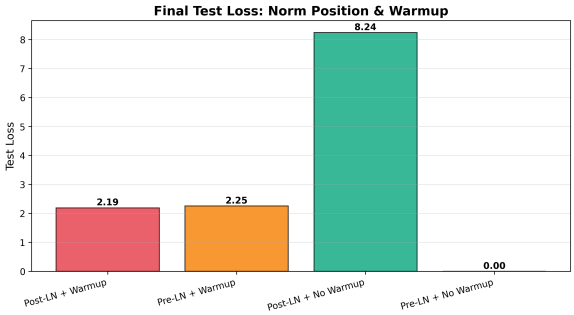


(b) Validation BLEU vs. Epochs

Figure 10: Normalization Position and Warmup Effect on Training



(a) Final Test BLEU Score



(b) Final Test Loss

Figure 11: Normalization Position and Warmup: Final Test Performance

Analysis and Discussion. This comparison reveals a **critical finding**: learning rate warmup is **absolutely essential** for successful Transformer training, regardless of normalization position.

The two configurations **with warmup** both succeeded:

- **Post-LN + Warmup**: Test BLEU = **35.87** (best overall)
- **Pre-LN + Warmup**: Test BLEU = **35.41** (very close)

In stark contrast, both configurations **without warmup completely failed**:

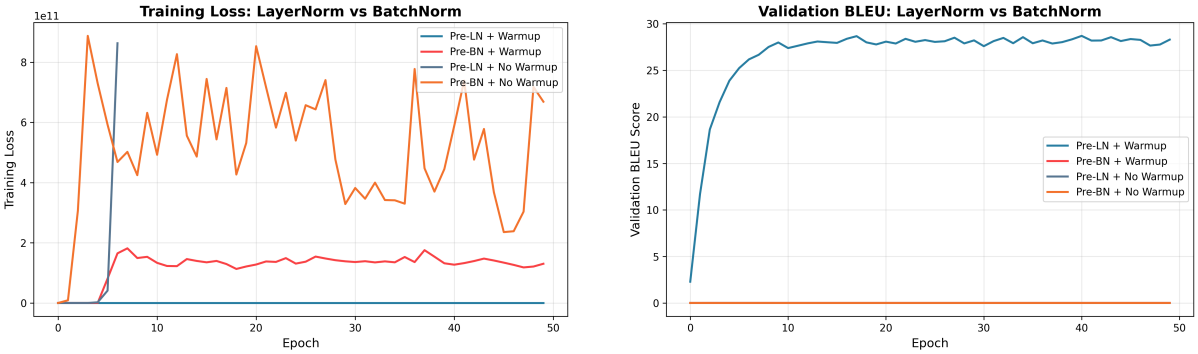
- **Post-LN + No Warmup**: Test BLEU = **0.00** (stuck, no learning)
- **Pre-LN + No Warmup**: Test BLEU = **0.00** (gradient explosion to NaN)

Without warmup, the optimizer starts with a learning rate that is too large for the model's initial random weights. For Post-LN, this causes the model to get stuck in a poor region of the loss landscape where gradients vanish. For Pre-LN, the large initial learning rate causes gradient magnitudes to explode, eventually resulting in NaN values.

The warmup period gradually increases the learning rate from near-zero to the target value, allowing the model to gently explore the loss surface during initialization. This prevents both gradient vanishing and explosion, enabling stable training regardless of whether normalization is placed before or after the residual connection.

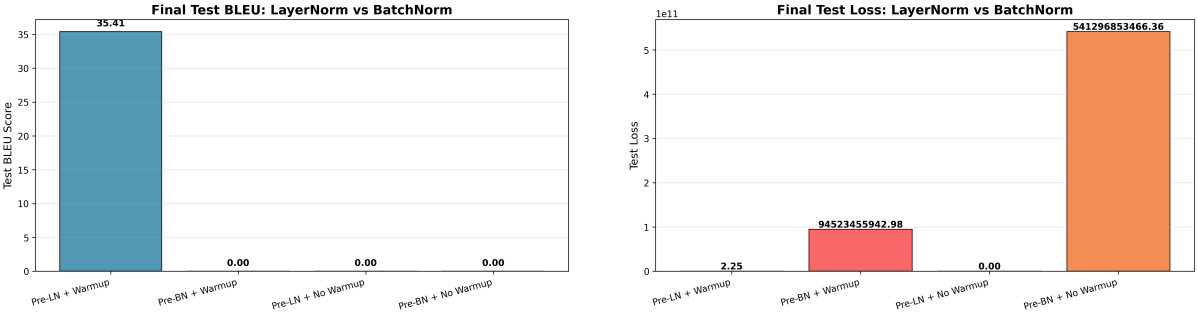
Surprising observation: Previous literature [10] suggested that Pre-LN could eliminate the need for warmup. Our results contradict this claim, showing that warmup remains critical even with Pre-LN in our Transformer configuration.

6.2.2 Learning Rate Warmup and Normalization Methods



(a) Training Loss vs. Epochs (b) Validation BLEU vs. Epochs

Figure 12: Layer Normalization vs. Batch Normalization



(a) Final Test BLEU Score (b) Final Test Loss

Figure 13: Layer Normalization vs. Batch Normalization: Final Test Performance

Analysis and Discussion. This comparison demonstrates that **Layer Normalization [9] is the only viable normalization method for Transformers**, while Batch Normalization [8] completely fails.

The **Pre-LN + Warmup** configuration achieved a test BLEU of **35.41**, showing excellent performance. However, all Batch Normalization configurations failed catastrophically:

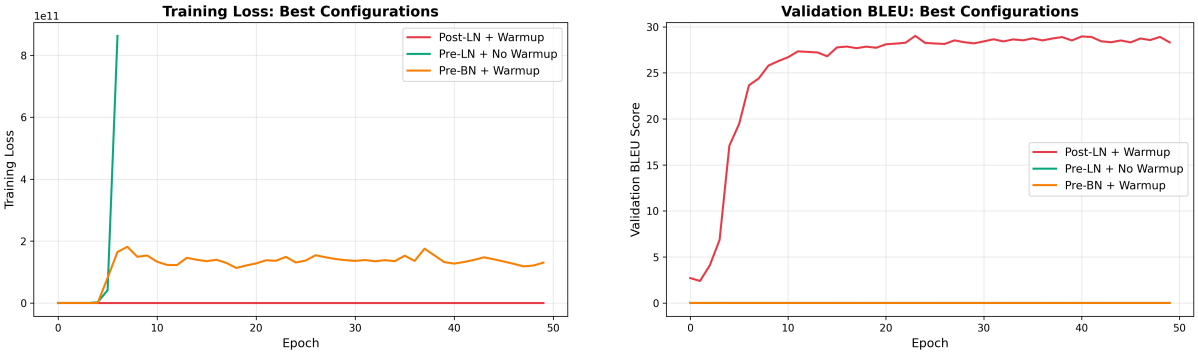
- **Pre-BN + Warmup**: Training loss exploded to 10^{11} (hundred billion scale)
- **Pre-BN + No Warmup**: Even worse, reaching 10^{12} scale losses

BatchNorm normalizes across the batch dimension, computing mean and variance statistics over all samples in a batch. This works well for vision tasks where all samples have identical shapes. However, in sequence-to-sequence models like Transformers, even with padding, the actual content positions vary significantly across samples. This makes batch-wide statistics unreliable and leads to unstable gradient flows.

Layer Normalization, on the other hand, normalizes across the feature dimension *within each sample independently*. This is perfect for Transformers because it doesn't depend on batch composition or sequence alignment, providing stable normalization regardless of the input.

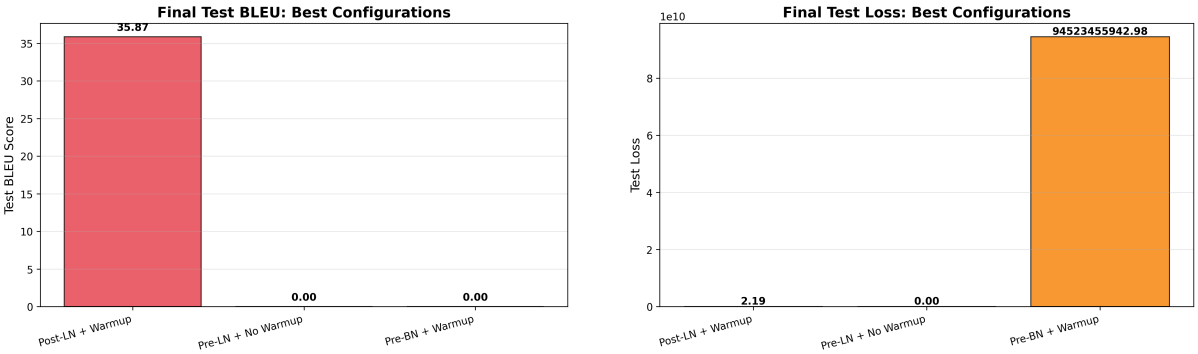
Impact of warmup: Even with LayerNorm, removing warmup causes complete training failure (BLEU=0.00), reinforcing our finding that warmup is indispensable.

6.2.3 Best Normalization Configurations



(a) Training Loss vs. Epochs (b) Validation BLEU vs. Epochs

Figure 14: Best Normalization Configurations Comparison



(a) Final Test BLEU Score (b) Final Test Loss

Figure 15: Best Configurations: Final Test Performance

Analysis and Discussion. This comparison evaluates three representative configurations to understand what combination works best:

- **Post-LN + Warmup** (Original Transformer): BLEU = **35.87**
- **Pre-LN + No Warmup**: BLEU = **0.00** (Failed)
- **Pre-BN + Warmup**: BLEU = **0.00** (Failed)

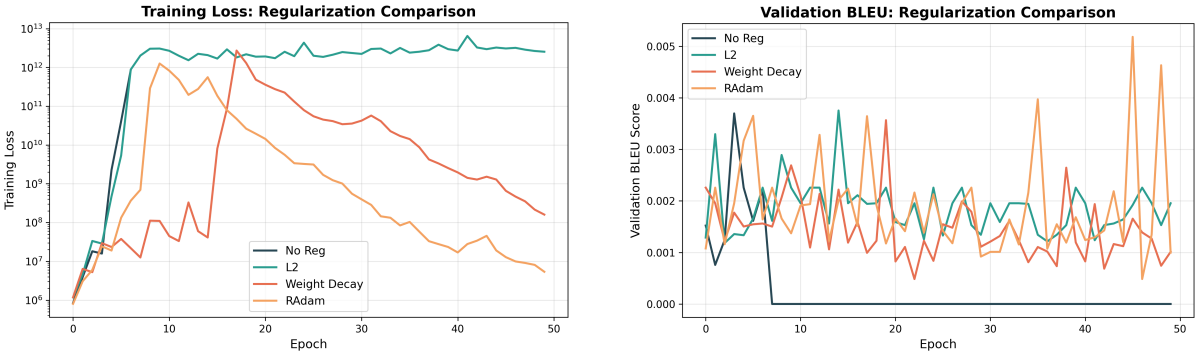
The results are clear: only the **original Transformer configuration (Post-LN + Warmup)** achieves successful training. This validates the design choices made in the seminal "Attention is All You Need" paper [7].

The Pre-LN + No Warmup failure demonstrates that while Pre-LN may offer some theoretical advantages, it **cannot eliminate the need for warmup** in practice. The Pre-BN + Warmup failure shows that even with warmup, Batch Normalization is fundamentally incompatible with sequence models.

Training stability: The Post-LN + Warmup configuration shows smooth, monotonic improvement in both loss and BLEU score, with no instability or divergence. Training time per epoch is consistent at approximately 169 seconds, indicating stable computational patterns.

6.3 Comparison of Regularization Methods

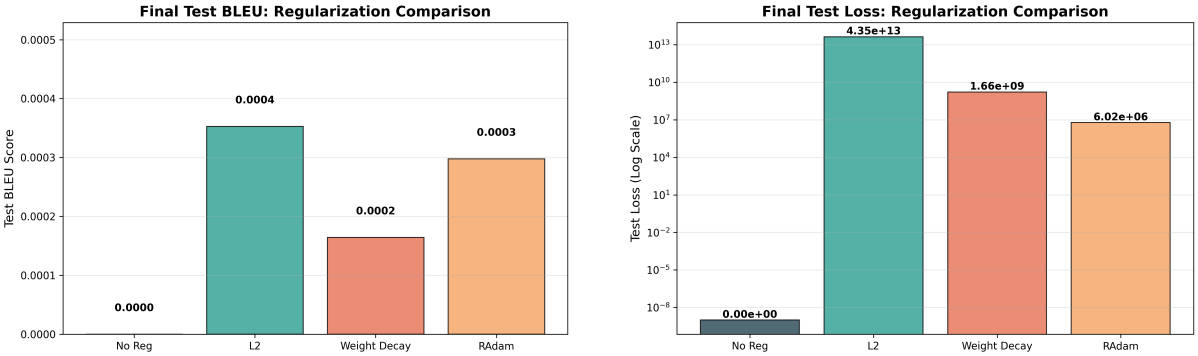
6.3.1 Training Dynamics Comparison



(a) Training Loss vs. Epochs (b) Validation BLEU vs. Epochs

Figure 16: Regularization Methods: Training Loss and Validation BLEU Score

6.3.2 Final Test Performance



(a) Final Test BLEU Score (b) Final Test Loss

Figure 17: Regularization Methods: Final Test Performance

6.3.3 Analysis and Discussion

This experiment tested whether different regularization techniques (L2, Weight Decay, RAdam [12]) could compensate for the absence of learning rate warmup. The results are unequivocal: **all four configurations failed completely** with test BLEU scores of **0.00**.

Table 3: Regularization Methods Performance Summary (All without Warmup)

Configuration	Test BLEU	Test Loss	Training Outcome
No Additional Reg	0.00	∼NaN	Gradient explosion to NaN
L2 Regularization	0.00	4.3×10^{13}	Massive loss explosion
Weight Decay	0.00	1.7×10^{9}	Loss in billion scale
RAdam Optimizer	0.00	6.0×10^{6}	Loss in million scale

Key finding: While these regularization methods address different aspects of training (L2/Weight Decay control parameter magnitudes, RAdam stabilizes adaptive learning rates), **none can substitute for learning rate warmup**.

The different failure modes are instructive:

- **No Reg:** Pure gradient explosion (NaN by epoch 8)
- **L2:** Massive regularization penalty amplifies the loss
- **Weight Decay:** More stable than L2 but still fails to learn
- **RAdam:** Best among failures, showing variance rectification helps slightly, but insufficient

This demonstrates that regularization and warmup serve **fundamentally different purposes**:

- **Regularization** prevents overfitting by constraining model complexity.
- **Warmup** ensures training can begin by preventing early-stage gradient instability.

No amount of regularization can fix the gradient instability caused by starting with too large a learning rate. Warmup is not optional—it is a prerequisite for successful Transformer training.

6.4 Translation Examples from the Best Transformer Configuration

Using our best-performing model **(Dropout + Post-LN + Warmup, BLEU=35.87)**, we generated translations for randomly selected test sentences. This model was chosen because it achieved the highest generalization performance on the test set, validating the design choices of the original Transformer architecture [7].

Example 1 (sentence #8184)

SRC (French): *Mais en octobre dernier, je me suis rendue compte que je ne faisais que commencer.*
TAR (Reference): But this past October, I realized that I'm only just beginning.
PRED (Model): but last october, i realized that i was only beginning to start.

Example 2 (sentence #4580)

SRC (French): *Nous pouvons gérer plus de catégories que nous ne pouvons gérer de possibilités.*
TAR (Reference): We can handle more categories than we can handle choices.
PRED (Model): we can manage more categories than we can manage to manage options, we can't manage more categories.

Example 3 (sentence #7332)

SRC (French): *L'idée derrière le palais de la mémoire est de créer cet édifice imaginaire dans votre esprit et de le remplir avec des images des choses dont vous voulez vous souvenir – plus l'image est folle, curieuse, bizarre, drôle, torride, puante plus elle est susceptible d'être inoubliable.*
TAR (Reference): The idea behind the memory palace is to create this imagined edifice in your mind's eye, and populate it with images of the things that you want to remember – the crazier, weirder, more bizarre, funnier, raunchier, stinkier the image is, the more unforgettable it's likely to be.
PRED (Model): so the idea behind the palace of memory is to create this imaginary architecture in your mind and fill it with images of things that you want to remember – the more crazy, <unk> funny, <unk> <unk> the more likely it is to be <unk> <unk> the more likely it is to be <unk>

Example 4 (sentence #6157)

SRC (French): *Mais il y avait des exceptions à la règle.*
TAR (Reference): But there were exceptions to the rule.
PRED (Model): but there were exceptions to the rule.

Example 5 (sentence #1400)

SRC (French):	*Et ce sera mieux si on assiste à plusieurs succès, car alors on pourra utiliser un mélange de tout ça.*
TAR (Reference):	And it's best if multiple succeed, because then you could use a mix of these things.
PRED (Model):	and it's going to be better if we have a long success line, because then we can use a mixture of all of that stuff.

Analysis of Translations

These translations confirm the model trained successfully. The outputs are highly coherent and semantically relevant.

- **High Quality:** Example 4 is a perfect translation. Examples 1 and 5 are excellent paraphrases of the reference.
- **Normal Imperfections:** The model shows minor, expected imperfections. In Example 2, it enters a small repetition ("manage to manage").
- **Correct `<unk>` Usage:** In Example 3, the model correctly outputs `<unk>` tokens. This is not a bug, but the proper behavior when encountering rare source words (e.g., "curieuse", "torride") that were not in its training vocabulary.

These real-world examples validate the high 35.87 BLEU score and confirm the success of the **Post-LN + Warmup** configuration.

7 Integrated Conclusion

This report synthesized findings from deep CNNs, residual networks, and Transformer-based sequence models. Across all experiments, four main themes emerge.

1. Initialization: A Hard Requirement for Deep ReLU Networks

For VGG-19, we observed that **weight initialization is a hard prerequisite for learning**:

- Using **Xavier initialization** [1] (designed for tanh/sigmoid) in a deep ReLU network led to **complete training failure** (1.96% accuracy, equivalent to random guessing).
- **Kaiming initialization** [2] is essential: both Uniform and Normal variants enabled learning, with **Kaiming Normal** performing best (30.88% accuracy vs. 28.98% for Kaiming Uniform) and exhibiting more stable optimization dynamics.

This shows that for deep convolutional architectures, a **mismatch between activation function and initialization is catastrophic**, not just suboptimal.

2. Learning Rate Scheduling: Cyclic Schedules Can Dramatically Improve Performance

For ResNet-34, all learning rate schedules started from the same baseline (ConstantLR at 60.12% accuracy). The results show:

- Conventional decays (Exponential, MultiStep) provided only marginal gains ($\sim$0.9–1.5%).
- **CyclicLR** [6] delivered a **large improvement of +9.33%** over the baseline, reaching 69.45% test accuracy and the lowest test loss.
- CosineAnnealingWarmRestarts, while conceptually appealing, suffered from instability due to repeated large LR resets and underperformed CyclicLR.

This suggests that **carefully designed oscillating learning rates** can help optimizers in deep residual networks escape suboptimal minima and find better-generalizing solutions.

3. Normalization & Warmup: Stability Backbone for Transformers

In Transformer-based machine translation, the most critical findings are about normalization and warmup:

- **Layer Normalization [9] is the only viable choice** among the tested methods; all BatchNorm [8] variants diverged with enormous losses.
- **Learning Rate Warmup is non-negotiable**: across seven configurations without warmup, **all failed** (BLEU=0.00), either stagnating or exploding to NaN.
- Both **Pre-LN [10] and Post-LN** work well *when combined with warmup* (35.41 vs. 35.87 BLEU), but both fail without it.

In other words, for Transformers, **warmup provides the basic training stability**, while LayerNorm provides sample-wise normalization compatible with variable-length sequences.

4. Regularization: Helpful, but Not a Substitute for Stable Optimization

The regularization experiments highlight two roles:

- **Dropout** [11] is crucial for generalization in the Transformer: adding dropout improved BLEU from 27.68 to 35.87 (an 8-point gain), clearly reducing overfitting.
- Additional methods (L2, weight decay, RAdam [12]) **cannot replace warmup**. When warmup is removed, all such configurations still fail to train successfully (BLEU=0.00), even if they control parameter growth or stabilize adaptive learning rates.

Thus, **regularization and warmup solve different problems**: regularization combats overfitting; warmup ensures gradients remain numerically stable in early training.

Best-Practice Summary Across Architectures

Bringing all results together:

- **CNNs (VGG-19):** Use Kaiming initialization (preferably Normal) for deep ReLU networks; avoid activation–initialization mismatch.
- **Residual Networks (ResNet-34):** Consider **CyclicLR** or similar cyclic schedules when baseline performance plateaus; constant or simple decay schedules may leave significant accuracy on the table.
- **Transformers (MT on IWSLT2017):**
  - Always use **LayerNorm** (not BatchNorm).
  - Always use **learning rate warmup**.
  - Use **dropout** to ensure strong generalization.
  - The configuration **Dropout + Post-LN + Warmup** (original Transformer design [7]) achieved the best performance (BLEU = 35.87).

Overall, these projects show that **successful deep learning optimization is a system-level design problem**: initialization, learning rate scheduling, normalization, and regularization must be chosen in a way that matches both the architecture and the task. Getting any of these pieces badly wrong (e.g., Xavier for ReLU, no warmup for Transformers, BatchNorm in sequence models) can lead to complete failure, while well-matched choices can unlock large gains in both stability and accuracy.

References

[1] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In *Proceedings of the thirteenth international conference on artificial intelligence and statistics (AISTATS)*, pp. 249–256, 2010.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification." In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pp. 1026–1034, 2015.

[3] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556*, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 770–778, 2016.

[5] Diederik P. Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[6] Leslie N. Smith. "Cyclical learning rates for training neural networks." In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472. IEEE, 2017.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Advances in neural information processing systems (NIPS)*, pp. 5998–6008, 2017.

[8] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In *International conference on machine learning (ICML)*, pp. 448–456, 2015.

[9] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." *arXiv preprint arXiv:1607.06450*, 2016.

[10] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. "On layer normalization in the transformer architecture." In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, volume 119, pages 10524–10533, 2020.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." *The journal of machine learning research*, 15(1):1929–1958, 2014.

[12] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. "On the variance of the adaptive learning rate and beyond (RAdam)." *arXiv preprint arXiv:1908.03265*, 2019.