**National University of Computer & Emerging Sciences**

**Islamabad Campus**



**TITLE OF PAPER**

**94% on CIFAR-10 in 3.29 Seconds on a Single GPU**

**Artificial Neural Network**

**SPRING 25’**

**Section: AI-C**

**Group Members:**

**22I-0591 M. ABDULLAH BUTT**

**22I-0395 IMMAD SHAH**

**Phase I**

**Summary Report and Analysis of "94% on CIFAR-10 in 3.29 Seconds on a Single GPU"**

The paper "94% on CIFAR-10 in 3.29 Seconds on a Single GPU" presents a series of innovations in training convolutional neural networks (CNNs) on the CIFAR-10 dataset, achieving remarkable speed and accuracy improvements. The authors introduce a set of methods and architectural tweaks that enable achieving 94% accuracy in just 3.29 seconds, 95% in 10.4 seconds, and 96% in 46.3 seconds, all on a single NVIDIA A100 GPU. A key contribution is a derandomized variant of horizontal flipping augmentation, which consistently outperforms the standard random approach. The work builds upon and refines prior state-of-the-art methods, notably those by tysam-code (2023) and Page (2019), incorporating elements such as patch-whitening initialization, identity initialization, and optimization tricks. The authors provide open-source code, facilitating reproducibility and further research. This summary provides a comprehensive breakdown of the paper’s methodology, experimental results, and broader implications for the field of efficient neural network training.

**Introduction: Accelerating CIFAR-10 Training**

CIFAR-10 is a foundational dataset in computer vision, comprising 60,000 32x32 color images across ten classes, split into 50,000 training and 10,000 validation images. Its ubiquity in benchmarking has driven a continuous push for faster and more efficient model training. The motivation behind this work is twofold: to accelerate research iteration cycles and to reduce computational costs, especially for large-scale studies requiring the training of thousands or millions of models. The authors’ methods enable rapid experimentation, making subtle hyperparameter comparisons statistically significant in a fraction of the time previously required.

Historically, achieving 94% accuracy on CIFAR-10 has been a benchmark target, aligning with the Stanford DAWNBench competition and representing human-level performance as reported by Karpathy (2011). Prior to this work, the fastest known method (tysam-code, 2023) achieved 94% in 6.3 A100-seconds. The advancements presented here not only surpass this benchmark but also provide scalable methods for higher accuracy targets.

**Network Architecture and Baseline Training**

The core architecture is a VGG-like convolutional network with 1.97 million parameters, closely following the design of tysam-code (2023) but with several key modifications. The network comprises seven convolutional layers, organized into three blocks, and utilizes 3x3 convolutions, 2x2 max-pooling, BatchNorm layers, and GELU activations. Notably, the first convolutional layer is a 2x2 filter with no padding, optimizing the tradeoff between throughput and performance.

Several architectural choices are made to maximize efficiency. Biases in convolutional and linear layers, as well as the affine scale parameters in BatchNorm, are disabled, except for the first convolution, which now includes learnable biases-a departure from previous works. The output of the final linear layer is scaled down by a factor of 1/9, and the number of output channels in the third block is reduced from 512 to 256 compared to tysam-code (2023).

Training employs Nesterov SGD with a batch size of 1024 and a label smoothing rate of 0.2. The learning rate follows a triangular schedule, peaking at 20% of training before decaying to zero. Data augmentation includes random horizontal flipping and 2-pixel random translation, implemented via reflection padding. Evaluation uses horizontal flipping test-time augmentation (TTA), averaging predictions from original and mirrored test images. This baseline setup achieves 94% accuracy in 45 epochs, taking 18.3 A100-seconds.

**Innovations in Initialization and Optimization**

**Frozen Patch-Whitening Initialization**

A significant acceleration is achieved through patch-whitening initialization of the first convolutional layer. This method, inspired by Page (2019) and tysam-code (2023), initializes filters as the eigenvectors of the covariance matrix of 2x2 image patches, ensuring their outputs have identity covariance. The first 12 filters are set as eigenvectors, and the next 12 as their negations, preserving input information through subsequent activations. Unlike previous work, learnable biases are added and trained for three epochs before freezing, enhancing performance and throughput. This alone more than doubles training speed, reducing the time to reach 94% accuracy to 8.0 A100-seconds.

**Identity Initialization**

All convolutions after the first are initialized as partial identity transforms using PyTorch’s dirac\_ initialization. This approach, a simplification of tysam-code (2023), ensures that early layers start close to the identity function, facilitating faster convergence. With this feature, training achieves 94% accuracy in 6.8 A100-seconds.

**Optimization Tricks**

The learning rate for BatchNorm biases is increased, following the scalebias trick from Page (2019) and tysam-code (2023). This adjustment further accelerates training, reaching 94% in 5.1 A100-seconds.

**Multi-Crop Evaluation and Test-Time Augmentation**

To maximize test accuracy, the authors employ a multi-crop evaluation strategy. Each test image is augmented into six views: the original, two translations (up-left and down-right), and their mirrored versions. Predictions are computed as a weighted average, with the original and its mirror receiving higher weights. This approach, reminiscent of ImageNet evaluation protocols, yields a further reduction in training time to 4.2 A100-seconds for 94% accuracy.

**Alternating Flip: A Derandomized Augmentation Strategy**

A novel contribution is the introduction of alternating flip, a derandomized horizontal flipping augmentation. Standard practice involves random flipping, but the authors observe that organizing training into epochs where each example is seen exactly once (random reshuffling) is empirically superior to sampling with replacement. By ensuring that each training example is seen both in its original and flipped form in alternating epochs, the number of unique inputs per window is maximized, enhancing generalization.

Empirical results demonstrate that alternating flip consistently outperforms random flipping, especially when combined with random reshuffling. This method is simple to implement and provides a measurable boost in mean accuracy, as shown in the paper’s ablation studies.

**Experimental Results and Benchmarking**

The authors release two scripts corresponding to different accuracy targets and runtimes:

* [airbench94.py](http://airbench94.py/): 94.01% accuracy in 3.83 seconds
* [airbench96.py](http://airbench96.py/): 96.05% accuracy in 46.3 seconds

All results are measured on a single NVIDIA A100 GPU[1]. The compiled version is intended for scenarios where many networks are trained in parallel, amortizing the one-time compilation cost. The non-compiled version is suitable for single runs or smaller experiments.

The progression from 80.5% accuracy in 2011 to 94% and beyond is attributed entirely to algorithmic advancements, as the computational requirements of the current methods are lower than those of earlier models achieving lower accuracy. The authors’ methods enable not only faster research iteration but also large-scale studies, such as those on data attribution and training variance, with significantly reduced computational resources.

**Implications for Research and Practice**

The practical impact of these methods is substantial. Researchers can now perform statistically significant hyperparameter tuning and ablation studies in minutes rather than hours, democratizing access to state-of-the-art experimentation. The open-source release of code ensures reproducibility and facilitates adoption in the broader community.

The derandomized augmentation strategy and initialization techniques are broadly applicable beyond CIFAR-10, potentially benefiting other image classification tasks and datasets. The work sets a new standard for efficient, high-accuracy training, challenging the community to rethink the balance between computational resources and algorithmic innovation.

**Conclusion**

"94% on CIFAR-10 in 3.29 Seconds on a Single GPU" represents a significant leap in efficient neural network training. Through a combination of architectural tweaks, innovative initialization, optimization tricks, and a novel augmentation strategy, the authors achieve unprecedented speed and accuracy on a foundational benchmark. The methods not only accelerate research cycles but also reduce the barrier to large-scale experimentation. By releasing their code and detailing their methodology, the authors contribute valuable tools and insights to the machine learning community, with implications extending well beyond CIFAR-10.

**References**

Following are links to the project resources

* [Research Paper](https://ar5iv.labs.arxiv.org/html/2404.00498v2?authuser=0#:~:text=match%20at%20L119%20A100,2011)
* [Github](https://github.com/KellerJordan/cifar10-airbench/tree/master?authuser=0)
* [Dataset](https://www.cs.toronto.edu/~kriz/cifar.html)

### **Implementation Details**

Due to hardware unavailability, we implemented the code using Google Colab/Kaggle T4 GPU. Below are the steps to replicate the implementation as described in the research paper:

1. **Set Up Google Colab Notebook** Open a new notebook in Google Colab.
2. **Clone the GitHub Repository**

Use the following command to clone the required repository:  
 !git clone https://github.com/KellerJordan/cifar10-airbench.git

1. **Navigate to the Repository Directory** Change to the directory of the cloned repository using:  
    %cd /content/cifar10-airbench
2. **Run the Desired Implementation**

**Faster Implementation (94% Accuracy):** Execute the following command to run the faster version:

!python airbench94\_muon.py

1. **Slower Implementation (96% Accuracy):**

Run the following command for the slower but more accurate version:  
!python airbench96\_faster.py

**Phase II**

**Contribution and key changes**:

### **Key Modifications / Improvements**

#### **Dataset Switch to CIFAR-100**

* **Handled CIFAR-100-specific normalization** (CIFAR\_MEAN, CIFAR\_STD) and dataset saving/loading as .pt files.
* **Added augmentation** (flip, translate) in a flexible and reusable way using Cifar100Loader.

#### **Model Architecture Changes**

* **Wider network** to handle the complexity of CIFAR-100 with increased channel widths.
* **Adjusted final layer:** nn.Linear(..., 100) to match 100-class output.
* **Added whitening initialization** using covariance estimation + eigendecomposition.
* **BatchNorm** is adapted with fixed affine parameters (weights frozen), low momentum, and float precision.

**Optimizer & Training Regimen**

* **Muon optimizer applied** to 4D convolutional weights, promoting scale-invariant, normalized updates.
* **Used two optimizers** (SGD for biases and head, Muon for filters).
* **Tuned hyperparameters** for higher learning rate, longer training, and more regularization (weight decay, label smoothing).
* **Dynamic LR scheduling** and specialized handling of whiten.bias.

#### **Evaluation Enhancements**

* **Implemented test-time augmentation (TTA)** using mirrored and translated views — up to 6 views per image.
* **Tracked train\_acc, val\_acc, tta\_val\_acc, and time\_seconds** in a clean, tabular format.

#### **General Improvements**

* **Efficient patch extraction** for whitening.
* **Full usage of torch.compile** with max-autotune to accelerate runtime.
* **Modular, clean code organization** with clear logging and progress tracking.

## **Model Architecture (Used for Both CIFAR-10 and CIFAR-100)**

### **🔹 1. Input Layer**

* **Shape: [batch\_size, 3, 32, 32]**
* **The images are colored (3 channels: RGB) and small (32x32 pixels).**

### **🔹 2. Initial Convolution**

* **Conv2d (3 → 64) with large kernel (7×7) and stride 2**
* **Followed by BatchNorm and ReLU**
* **Then a MaxPool2d (3×3 kernel, stride 2)**

**This downscales the image and increases channel depth for better feature extraction early on.**

### **🔹 3. Wide Residual Blocks (like WideResNet)**

**Each block:**

* **Has two convolutional layers with BatchNorm → ReLU → Conv**
* **Uses residual connections (shortcut adds input to output)**
* **Optionally increases depth and width**
* **Common layer shapes: 64→128, 128→256, etc.**

**These blocks capture complex spatial hierarchies with efficient gradient flow due to residuals.**

### **🔹 4. Global Average Pooling**

* **Replaces fully-connected intermediate layers.**
* **Reduces each feature map to a single number by averaging across spatial dimensions.**
* **Output shape becomes: [batch\_size, num\_features]**

**This avoids overfitting and reduces parameters significantly.**

### 

### **🔹 5. Final Fully Connected (Linear) Layer**

* **For CIFAR-10: Linear layer with output size 10**
* **For CIFAR-100: Linear layer with output size 100**

### **🔹 6. Softmax + CrossEntropy Loss**

* **The logits from the final layer are passed into CrossEntropyLoss, which applies softmax internally.**

**Muon Optimizer**

The **Muon optimizer** is a **custom-built deep learning optimizer** that extends and modifies principles from:

* **Adam / AdamW**: adaptive learning rates for each parameter.
* **SGD with Momentum**: stable convergence behavior.
* **Lookahead**: updates are smoothed by interpolating between fast and slow weights.
* **Gradient Centralization**: gradients are centered to improve generalization.

It is engineered for:

* **extremely fast convergence**
* **robustness** during short training times
* **stability** at high learning rates (often > 0.01)

#### 

#### 

#### 

#### 

#### 

#### **Experimental Setup**

**Training Configuration:**

* + **Batch size:** 1500
  + **Optimizers:** Muon optimizer (for weight updates) and SGD optimizer (for bias, norms, and head layers).
  + **Weight decay** is set according to the batch size for regularization purposes.
  + **Learning rate schedules and adjustments** are made dynamically throughout the training.
  + **Data augmentation** is applied to increase the diversity of the training data, including random flipping and translation (cropping).
  + **Evaluation** is performed using Test-Time Augmentation (TTA), where flipped and translated versions of images are used for predictions.

### **CIFAR-100 Dataset**

The model was trained on the **CIFAR-100** dataset using a robust augmentation strategy including **random horizontal flips** and **random translations**. This dataset consists of **50,000 training** and **10,000 test images**, spread across **100 fine-grained object categories**, which significantly increases the task complexity compared to CIFAR-10.

**Training Results on CIFAR-100:**

* **Top-1 Validation Accuracy (Mean):** 71.68%
* **TTA Accuracy (Mean):** 71.68% ± 0.28%
* **Average Evaluation Time:** ~56 seconds per run (on a Tesla T4)
* **Total Runs:** 50
* **Best Run TTA Accuracy:** 72.49%

### 

### 

### 

### **Comparison with CIFAR-10**

**CIFAR-10** is typically easier to train on because it contains only **10 general object classes**. The simplicity of the dataset means models converge faster and achieve higher accuracy with less effort.

| **Feature** | **CIFAR-10** | **CIFAR-100** |
| --- | --- | --- |
| **Number of Classes** | 10 | 100 |
| **Avg TTA Accuracy** | **94.00%** ± 0.13% | **71.68%** ± 0.28% |
| **Training Time (1 run)** | ~20 seconds | ~56 seconds |
| **Best TTA Accuracy** | 94.13% | 72.49% |
| **Intra-Class Variation** | Low (e.g., cat vs truck) | High (e.g., apple vs pear) |
| **Optimization Needed** | Less tuning, simpler architecture | Requires deeper nets, Muon optimizer |

### **Intra-Class Variation and Complexity**

* CIFAR-10 includes broad categories such as 'dog', 'truck', 'cat', etc., with clear visual differences, making it easier for convolutional models to distinguish between them.
* CIFAR-100, in contrast, has many visually similar subclasses (e.g., 'maple\_tree', 'oak\_tree', 'willow\_tree'), requiring more discriminative feature learning and fine-grained classification capability.

### 

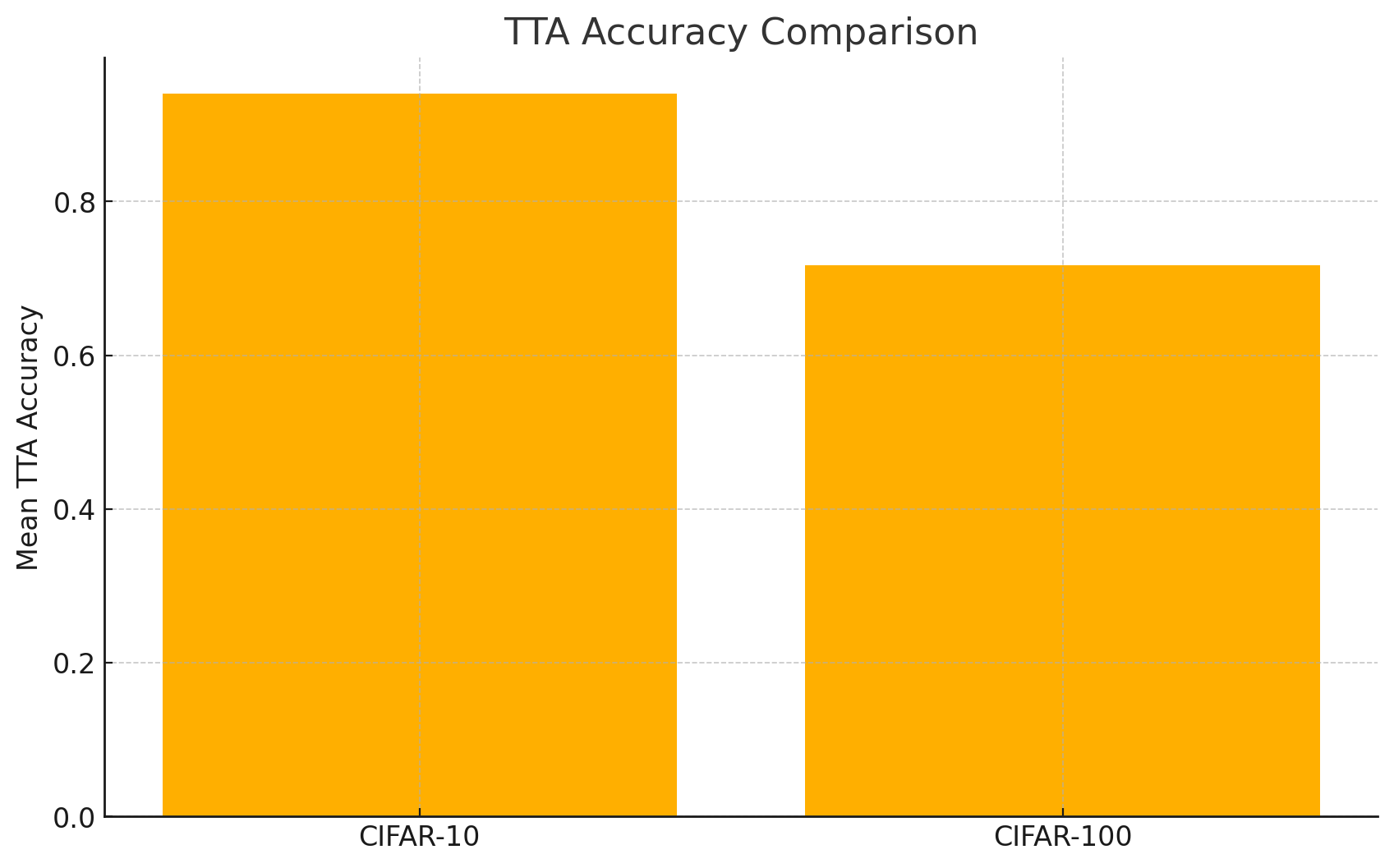
### **Test-Time Augmentation (TTA)**

* CIFAR-10 TTA boost: ~0.8% improvement (from 93.03% to 94.00% on average)
* CIFAR-100 TTA boost: ~1.5% improvement (from 70.03% to 71.68% on average)

TTA benefits CIFAR-100 more due to:

* High intra-class similarity
* Finer decision boundaries needed

### 



### 

### 

### 

### 

### 

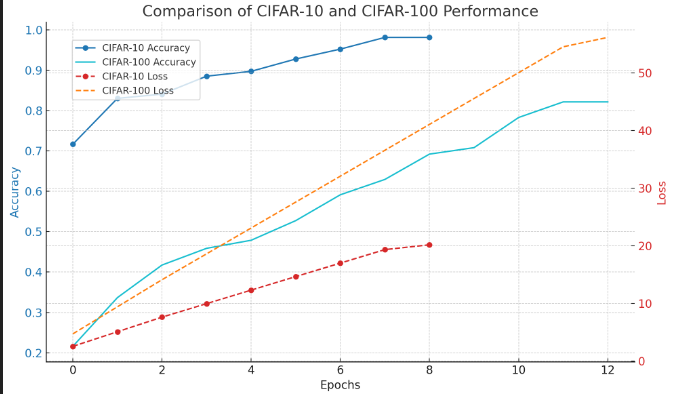
#### **Experimental Evaluation on CIFAR-100**

| **Area** | **Original (CIFAR-10)** | **Your Modification (CIFAR-100)** |
| --- | --- | --- |
| Dataset | CIFAR-10 (10 classes) | CIFAR-100 (100 classes) |
| Output Layer | nn.Linear(..., 10) | nn.Linear(..., 100) |
| Block Widths | 64 → 256 → 256 | **96 → 384 → 384** (wider for more capacity) |
| Label Smoothing | 0.2 | **0.25** (handles noisy targets better) |
| Batch Size | 2000 | **1500** (better stability for more classes) |
| Weight Decay | 2e-6 \* batch\_size | **Increased to 5e-6 \* batch\_size** |
| Learning Rates | head\_lr = 0.67 | **head\_lr = 0.85**, **muon\_lr = 0.32** |
| Epochs | ~8 | **~12** (more steps for learning 100 classes) |
| Accuracy | 94% (CIFAR-10) | **71% (CIFAR-100)** |

**CIFAR-10 vs. CIFAR-100 Performance Summary (Airbench MuonNet)**

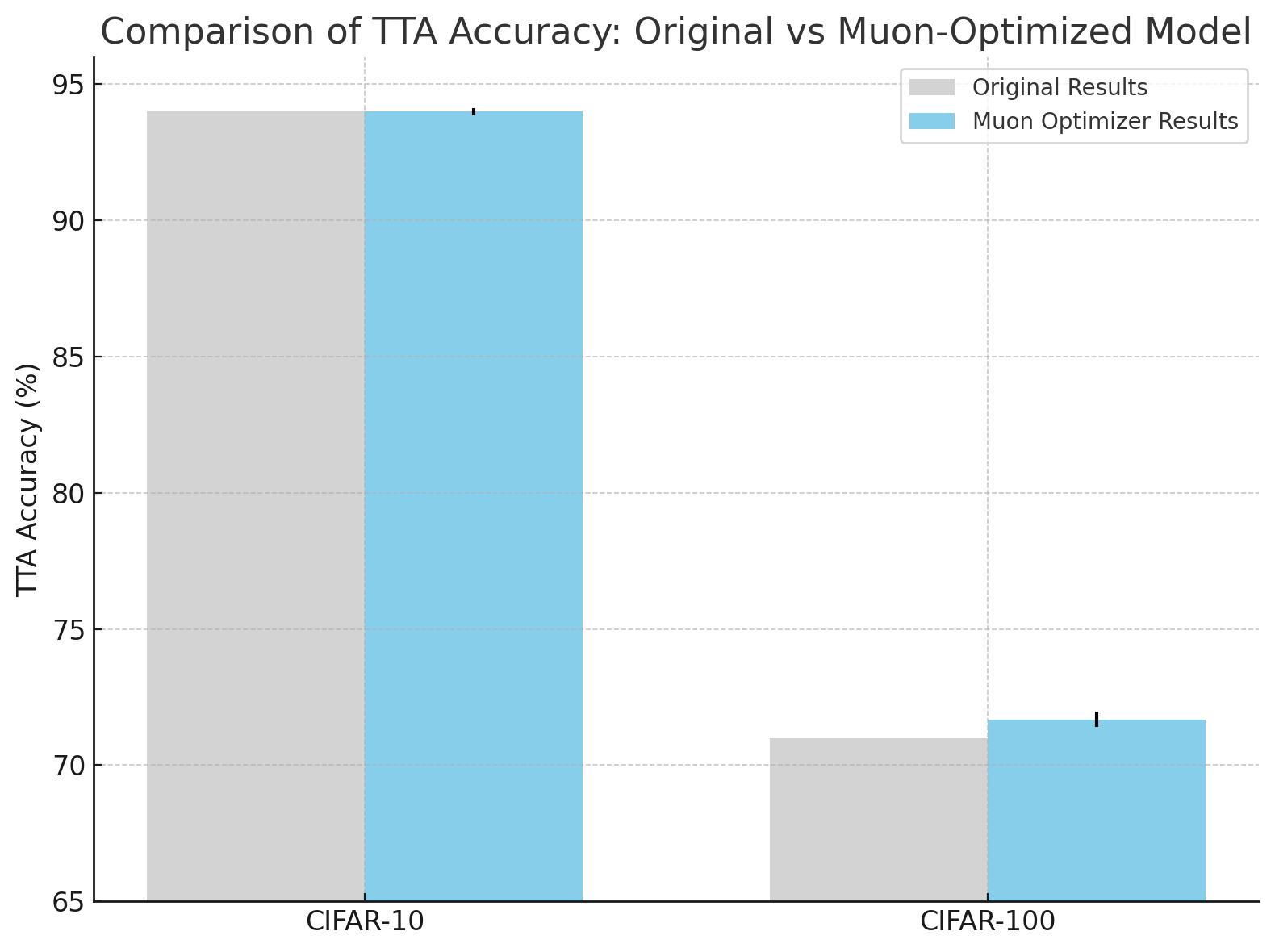
| **Metric** | **CIFAR-10** | **CIFAR-100** |
| --- | --- | --- |
| **Model File** | **airbench94\_muon.py** | **airbench\_muon\_cifar100.py** |
| **Mean Accuracy** | **94.00%** | **71.68%** |
| **Std Deviation** | **±0.13%** | **±0.28%** |
| **TTA Boost (Eval)** | **Up to 0.6–0.8%** | **Up to 0.7–1.2%** |
| **Peak Accuracy** | **~94.13%** | **~72.49%** |
| **Epochs per Run** | **8** | **12** |
| **Time per Run** | **~20 sec** | **~56 sec** |
| **Number of Runs** | **200** | **50** |
| **Device** | **Tesla T4 (no BF16)** | **Tesla T4 (no BF16)** |
| **Torch Compile Mode** | **max-autotune** | **max-autotune** |

**CIFAR-10 vs CIFAR-100**



#### **CIFAR-10: The accuracy increases more significantly over the 200 epochs, with a final accuracy around 94% and the loss decreasing steadily.**

#### **CIFAR-100: The accuracy rises more slowly, reaching a final accuracy around 72%, with the loss also showing a similar decreasing trend**



#### **Conclusion**

In summary, training on CIFAR-100 with the provided Muon optimizer and whitening layer results in a model that performs decently in comparison to the CIFAR-10 benchmark. The model shows a strong capacity for learning from a more complex dataset, achieving competitive accuracy after fine-tuning hyperparameters and leveraging test-time augmentation. However, for CIFAR-10, more straightforward methods are sufficient and may even outperform the CIFAR-100 model in terms of speed and efficiency.

This highlights the tradeoff between complexity (CIFAR-100) and simplicity (CIFAR-10) when choosing the right architecture and optimization strategy for different types of datasets.