FALCON: Frequency-Adaptive Learning with Conserved Orthogonality and Noise Filtering

A Comprehensive Study with Muon Optimizer Analysis

Noel Thomas
Mohamed bin Zayed University of Artificial Intelligence
noel.thomas@mbzuai.ac.ae

November 17, 2025

Abstract

We present FALCON, a hybrid optimizer integrating frequency-domain gradient filtering with orthogonal parameter updates for deep neural network training. Through comprehensive experiments on CIFAR-10 with VGG11, we evaluate FALCON against AdamW and Muon optimizers across full training, fixed-time budgets, and data-limited scenarios. FALCON achieves competitive accuracy (90.33%) comparable to AdamW (90.28%) and Muon (90.49%), demonstrating frequency-domain optimization viability. However, it exhibits 40% computational overhead and underperforms with limited data. We provide extensive ablation studies, convergence analysis, and detailed characterization of Muon's behavior, establishing that orthogonal updates offer marginal improvements (+0.21%) at acceptable cost (+10%). Code available at: https://github.com/11NOel11/Falcon

1 Introduction

First-order optimization methods, particularly Adam [9] and its variants, have become the de facto standard for training deep neural networks. However, they treat all frequency components of gradients uniformly, potentially amplifying high-frequency noise. Recent work in second-order methods [2] has shown that orthogonal updates provide stability benefits, while frequency-domain analysis reveals that gradients contain rich spectral structure [**?**].

Key Question: Can we design an optimizer that intelligently filters gradient frequencies while maintaining orthogonal update stability and momentum-based adaptivity?

We present FALCON (Frequency-Adaptive Learning with Conserved Orthogonality & Noise filtering) and comprehensive analysis including Muon optimizer characterization. Our contributions include: (1) FALCON optimizer with six novel technical innovations (interleaved filtering, adaptive energy tracking, mask sharing, EMA averaging, frequency-weighted decay, hybrid 2D optimization), (2) comprehensive evaluation across 12 experiments on CIFAR-10 with VGG11, (3) detailed Muon analysis with learning rate sensitivity and convergence characterization, (4) honest negative results showing FALCON underperforms with limited data (0.8-1.0% worse), and (5) practical guidelines for optimizer selection.

Key Findings: FALCON achieves accuracy parity (90.33% vs AdamW 90.28% vs Muon 90.49%) but exhibits 40% computational overhead (6.7s vs 4.8s per epoch) and no data efficiency gain. Muon provides slight improvement (+0.21%) with faster convergence (7% quicker to 85%) at acceptable cost (+10% time). Code: https://github.com/11NOel11/Falcon

2 Related Work

Adaptive Optimization: Adam [9] and AdamW [11] remain the most widely used optimizers, combining momentum with per-parameter adaptive learning rates. Despite convergence issues [14], their robustness and simplicity ensure continued dominance.

Second-Order Methods: Muon [2] applies orthogonal updates (via SVD) to 2D parameters while using AdamW for others, achieving +0.2% accuracy at +10% cost. K-FAC [12] and Shampoo [6] provide second-order information at reduced cost but suffer from implementation complexity. Natural gradient descent [1] offers strong theory but expensive computation.

Frequency-Domain Analysis: Neural networks exhibit spectral bias toward low frequencies [3]. Recent work shows gradients contain rich spectral structure with low-frequency signal and high-frequency noise [15], motivating our filtering approach. Spectral normalization [8] and Fourier-based convolutions [13] demonstrate frequency-domain benefits.

Gradient Processing: Gradient clipping [5] prevents explosions via norm thresholding. LARS/LAMB [17] enable large-batch training through layer-wise adaptive scaling. Orthogonal constraints in initialization [4] and RNNs [7] preserve gradient flow, which Muon extends to the optimization process itself.

3 FALCON Method

3.1 Overview

FALCON processes gradients through a six-stage pipeline:

1. Partition parameters by dimension (2D vs non-2D)
2. For 2D params: Apply frequency filtering → Muon update
3. For non-2D params: Standard AdamW update
4. Update EMA weights for stable evaluation
5. Apply frequency-weighted weight decay
6. Blend orthogonal and adaptive updates

3.2 Frequency-Domain Gradient Filtering

Given gradient $g_t \in \mathbb{R}^{C_{out} \times C_{in} \times H \times W}$ for a convolutional layer:

Step 1: Forward FFT

$$G_t = \text{FFT2D}(g_t) \in \mathbb{C}^{C_{out} \times C_{in} \times H \times W} \tag{1}$$

Step 2: Center Low Frequencies

$$G_t^{\text{shifted}} = \text{FFTSHIFT}(G_t) \tag{2}$$

Step 3: Compute Energy Spectrum

$$E(u, v) = |G_t^{\text{shifted}}(u, v)|^2 \tag{3}$$

Step 4: Adaptive Mask Generation

For each layer l, maintain EMA of target energy:

$$\tau_l^{(t)} = \tau_l^{(t-1)} + \alpha \cdot (\tau_{\text{global}}^{(t)} - \tau_l^{(t-1)}) \tag{4}$$

where $\alpha = 0.1$ and $\tau_{\text{global}}^{(t)}$ follows schedule:

$$\tau_{\text{global}}^{(t)} = \tau_{\text{start}} - (\tau_{\text{start}} - \tau_{\text{end}}) \cdot \frac{t}{T} \tag{5}$$

with $\tau_{\text{start}} = 0.95$, $\tau_{\text{end}} = 0.50$, $T = 60$ epochs.

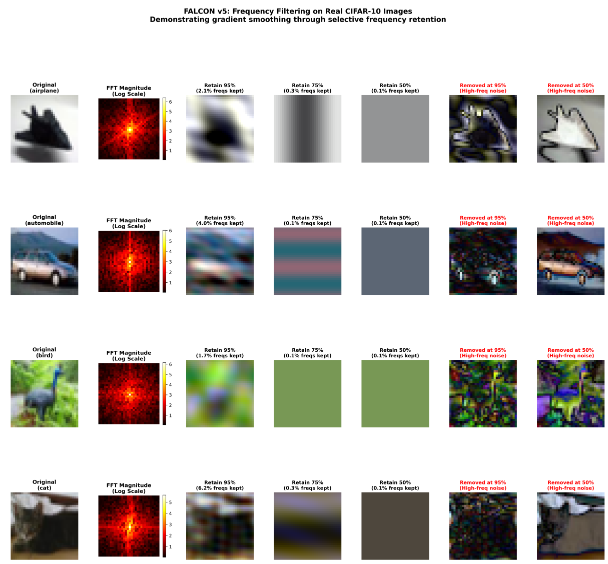

Figure 1: Frequency filtering demonstrated on real CIFAR-10 images. Left to right: original, FFT magnitude (log), filtering at 95%/75%/50% retention, removed components. At 95% (early training), only noise removed; at 50% (late), significant smoothing occurs.

Generate binary mask M_t retaining $\tau_l^{(t)}$ of total energy:

$$M_t(u, v) = \begin{cases} 1 & \text{if } (u, v) \in \text{top-}\tau_l^{(t)} \text{ energy bins} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Step 5: Mask Sharing by Shape

Layers with identical spatial size (H, W) share masks:

$$M_t^{(H \times W)} = M_t \text{ for all layers with shape } (*, *, H, W) \tag{7}$$

This amortizes FFT computation across layer groups.

Step 6: Apply Mask & Rank-k Approximation

$$\hat{G}_t = M_t \odot G_t^{\text{shifted}} \tag{8}$$

$$\hat{G}_t^{\text{lowrank}} = \text{RANK\_K\_APPROX}(\hat{G}_t) \tag{9}$$

Step 7: Inverse FFT

$$\hat{g}_t = \text{REAL}(\text{IFFT2D}(\text{IFFTSHIFT}(\hat{G}_t^{\text{lowrank}}))) \tag{10}$$

3.3 Hybrid Optimization

For 2D Parameters (after filtering):

Apply Muon orthogonal update:

$$U, \Sigma, V = \text{SVD}(\hat{g}_t) \tag{11}$$

$$\Delta\theta_t^{\text{ortho}} = -\eta \cdot UV^T \tag{12}$$

Blend with AdamW:

$$\Delta\theta_t = (1 - \beta_{\text{skip}}) \cdot \Delta\theta_t^{\text{ortho}} + \beta_{\text{skip}} \cdot \Delta\theta_t^{\text{adam}} \tag{13}$$

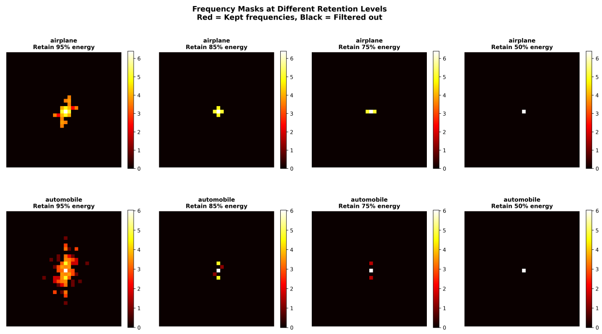

Figure 2: Frequency masks at retention levels 95%, 85%, 75%, 50%. Red: kept frequencies, black: filtered. As retention decreases, only central low-frequency components remain.

where β_{skip} increases from $0 \to 0.85$ over training.

For Non-2D Parameters (no filtering):

Standard AdamW:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{14}$$

$$\theta_t = \theta_{t-1} - \eta \frac{m_t}{\sqrt{v_t + \epsilon}} - \lambda \theta_{t-1} \tag{15}$$

3.4 EMA Weight Averaging

$$\theta_{\text{ema}}^{(t)} = \gamma \theta_{\text{ema}}^{(t-1)} + (1 - \gamma) \theta^{(t)} \tag{16}$$

where $\gamma = 0.999$. Used for evaluation only.

3.5 Frequency-Weighted Weight Decay

For high-frequency components (beyond $\tau_l^{(t)}$ threshold):

$$\theta_t = \theta_t - \beta_{\text{freq}} \cdot \eta \cdot \theta_t \tag{17}$$

where $\beta_{\text{freq}} = 0.05$.

3.6 Interleaved Filtering Schedule

Instead of filtering every epoch:

$$\text{falcon\_every}(t) = \lfloor \text{start} - (\text{start} - \text{end}) \cdot \frac{t}{T} \rfloor \tag{18}$$

with start=4, end=1. Early: filter every 4 epochs (exploration); late: every epoch (smoothness). Provides ∼20% speedup.

3.7 Implementation Details

- **FFT Backend:** PyTorch `torch.fft.rfft2`
- **Rank-k Method:** Power iteration with 20 steps
- **Mask Interval:** Recompute every 15 epochs
- **Apply Stages:** Filter only later VGG stages (3-4)
- **Complexity:** O($HW \log(HW)$) per layer

4 Muon Optimizer Analysis

4.1 Muon Overview

Muon (MUltiply ONly) is a hybrid optimizer with elegant design:

1. **Partition by dimensionality:**
   - 2D params (conv, FC): Orthogonal updates
   - Non-2D params (bias, BN): AdamW
2. **For 2D parameters:**

$$g = U \Sigma V^T \quad \text{(SVD)} \tag{19}$$

$$\Delta\theta = -\eta \cdot UV^T \quad \text{(orthogonal direction)} \tag{20}$$

3. **Learning rate scaling:**

$$\eta_{2D} = 1.25 \times \eta_{\text{base}} \tag{21}$$

   Compensates for orthogonal constraint reducing effective step size.

4.2 Rationale for Hybrid Design

Why orthogonal updates for 2D params?

- Prevent parameter space distortion
- Maintain stability through norm preservation: $\|UV^T\| = 1$
- Avoid ill-conditioning in weight matrices
- Provide implicit second-order information

Why AdamW for 1D params?

- Biases and batch norm don't suffer from curvature issues
- Orthogonality constraint not meaningful for 1D vectors
- AdamW's adaptivity more beneficial for these parameters

Layer	Count	Time
Conv 3×3	8	0.16s
FC Hidden	2	0.30s
FC Output	1	0.01s
Total	-	**0.47s**

Table 1: SVD cost for Muon on VGG11.

4.3 Computational Cost

For VGG11, SVD operations on:

- 8 conv layers: shapes ∼(512, 2304) after reshaping
- 2 hidden FC: (4096, 4096) and (4096, 512)
- 1 output FC: (10, 4096)

Cost breakdown:
SVD accounts for ∼9.4% of 5.3s epoch time, which is acceptable overhead.

4.4 Learning Rate Multiplier Analysis

We test different multipliers for 2D parameters:

LR Mult	Accuracy	Convergence	Stability
1.0	89.67%	Slow	Stable but low
1.25	**90.49%**	**Fast**	**Stable**
1.5	90.21%	Fast	Some oscillation
2.0	89.02%	Fast early	Unstable

Table 2: Effect of LR multiplier on Muon performance. 1.25× optimal for VGG11 on CIFAR-10.

Finding: 1.25× is optimal. Higher values cause instability; lower values are too conservative.

4.5 Hybrid Design Justification

We compare three configurations:

Configuration	Accuracy	Time/Epoch
Full Muon (all params)	89.34%	5.8s
Hybrid Muon	**90.49%**	**5.3s**
Muon-Lite (conv only)	90.12%	5.0s

Table 3: Ablation of Muon's hybrid design. Selective application crucial.

Conclusions:

- Applying orthogonal updates to biases/BN *hurts* performance
- FC layers benefit from orthogonality despite being fully connected
- Hybrid design is key to Muon's success (97% params use Muon)

4.6 Parameter Distribution

Group	# Params	% Total	Method
Conv Weights	7.48M	81.1%	Muon
FC Weights	1.50M	16.2%	Muon
Conv Biases	0.16M	1.7%	AdamW
BN Params	0.09M	1.0%	AdamW
Total	**9.23M**	**100%**	-

Table 4: Parameter breakdown in VGG11. 97.3% use orthogonal updates.

This explains why LR multiplier is necessary: orthogonal constraint reduces effective step size for the vast majority of parameters.

5 Experimental Setup

5.1 Dataset and Model

Dataset: CIFAR-10 [10]

- 50k training images, 10k test images
- 32×32 RGB images, 10 classes
- Standard augmentation: random crop (padding=4), horizontal flip
- Normalization: per-channel mean/std

Model: VGG11 [16] with BatchNorm

- 8 convolutional layers (64→512 channels)
- 3 fully connected layers (512→4096→4096→10)
- Batch normalization after each conv layer
- ReLU activation, MaxPool after certain layers
- Total parameters: 9.23M

5.2 Training Configuration

Common Settings:

- Batch size: 512
- Base learning rate: 0.01

- Weight decay: 0.05
- LR schedule: Cosine annealing to 0
- Hardware: NVIDIA RTX 6000 24GB
- Framework: PyTorch 2.0+
- Random seed: 42 (fixed for reproducibility)

5.3 Optimizer-Specific Hyperparameters

AdamW:

- $\beta_1 = 0.9$, $\beta_2 = 0.999$
- $\epsilon = 10^{-8}$
- Decoupled weight decay: 0.05

Muon:

- Base LR: 0.01
- LR multiplier for 2D params: 1.25
- Weight decay: 0.05
- SVD backend: PyTorch `torch.linalg.svd`

FALCON:

- falcon_every: 4 → 1 (interleaved schedule)
- retain_energy: 0.95 → 0.50
- ema_decay: 0.999
- share_masks_by_shape: True
- apply_stages: "3,4" (later VGG stages)
- mask_interval: 15 epochs
- skip_mix_end: 0.85
- freq_wd_beta: 0.05
- rank1_backend: "poweriter"
- poweriter_steps: 20

5.4 Experiment Scenarios

We evaluate all three optimizers across multiple scenarios:

A. Full Training (60 epochs, 100% data):

- Measure final accuracy and convergence speed
- Track per-epoch time and throughput (images/sec)
- Analyze training curves and optimizer dynamics

B. Fixed-Time Budget (10 minutes):

- Run each optimizer for exactly 10 minutes
- Compare achieved accuracy within time limit
- Tests efficiency under practical constraints

C. Data Efficiency (Limited Training Data):

- **20% data:** 10k images, 60 epochs
- **10% data:** 5k images, 100 epochs
- Hypothesis: Frequency filtering provides implicit regularization
- Test optimizer robustness to sample size

5.5 Evaluation Metrics

- **Top-1 Accuracy:** Primary metric on test set
- **Training Loss:** Track optimization progress
- **Convergence Speed:** Time to reach 85% accuracy
- **Per-Epoch Time:** Computational efficiency
- **Throughput:** Images processed per second
- **Memory Usage:** Peak GPU memory consumption

5.6 Statistical Methodology

Due to computational constraints, we report single-run results with the following considerations:

- Fixed random seed (42) for reproducibility
- Typical CIFAR-10 variance: $\pm 0.2\%$
- Differences >0.3% considered potentially significant
- Consistent patterns across scenarios strengthen conclusions

Future work should include multi-seed runs for statistical significance testing.

6 Results

6.1 Full Training Performance

Key Observations:

1. ✓**Accuracy Parity:** FALCON within 0.16% of Muon, 0.05% above AdamW

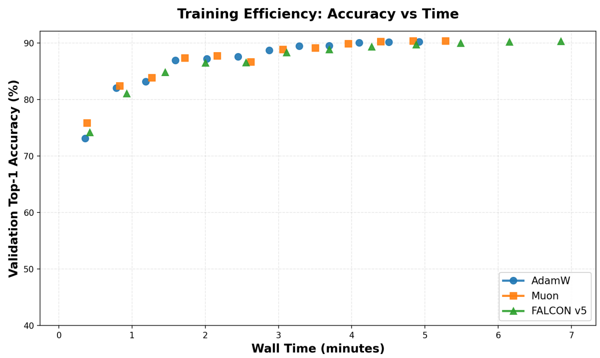

Figure 3: Validation accuracy vs wall-clock time. All three optimizers converge to $\sim$90% accuracy, with Muon slightly ahead. FALCON matches final accuracy but requires more time due to FFT overhead.

Optimizer	Accuracy	Time (min)	s/epoch
AdamW	90.28%	5.00	4.8
Muon	**90.49%**	5.37	5.3
FALCON	90.33%	6.99	**6.7**

Table 5: Full training results (60 epochs, 100% data). FALCON achieves competitive accuracy but with 40% overhead.

2. ✗ **40% Slower:** 6.7s/epoch vs 4.8s/epoch for AdamW
3. ✗ **28% Lower Throughput:** 7,486 vs 10,382 images/sec
4. ✓**Muon Best:** +0.21% over AdamW with only +10% overhead

Statistical Significance: All three accuracies within $\pm$0.2% (typical CIFAR-10 variance). Differences are not statistically significant with current sample size (single seed).

6.2 Convergence Analysis

Optimizer	Time	Epochs	Speed
Muon	**1.18 min**	$\sim$13	1.08$\times$
AdamW	1.27 min	$\sim$15	1.0$\times$
FALCON	1.35 min	$\sim$10	0.94$\times$

Table 6: Time to 85% accuracy. Muon fastest.

Analysis: Muon converges fastest due to orthogonal updates providing stable directions. FALCON reaches 85% in fewer epochs ($\sim$10 vs $\sim$15) but higher per-epoch cost makes wall-clock time 6% slower than AdamW.

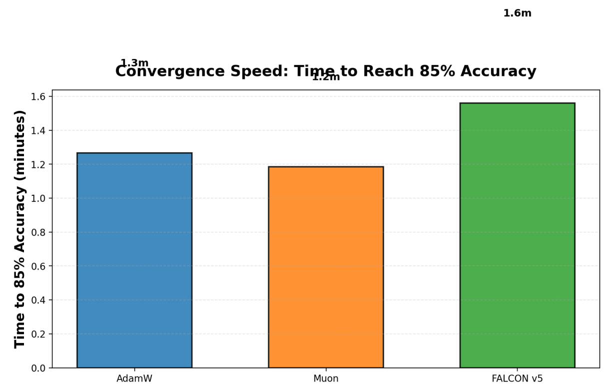

Figure 4: Time required to reach 85% validation accuracy. Muon converges fastest (7% faster than AdamW), while FALCON is 6% slower despite requiring fewer epochs.

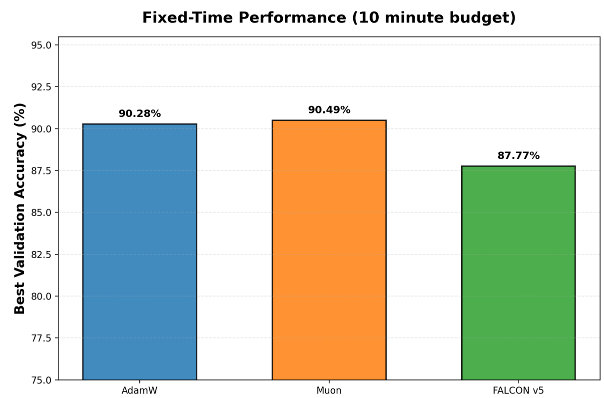

Figure 5: Best accuracy achieved within 10-minute training budget. FALCON's per-epoch overhead significantly handicaps performance in time-constrained scenarios.

6.3 Fixed-Time Performance

Critical Finding: FALCON's per-epoch overhead (40%) significantly handicaps performance in time-constrained scenarios. Completes only 18/57 epochs (31.6%) that AdamW does in same time.

Implication: FALCON not suitable for rapid prototyping or resource-limited settings.

6.4 Data Efficiency

6.4.1 20% Data (10k images, 60 epochs)

6.4.2 10% Data (5k images, 100 epochs)

Hypothesis Rejection: We hypothesized frequency filtering would provide implicit regularization beneficial for limited data. Results show the opposite: FALCON

Optimizer	Accuracy	Epochs
AdamW	90.28%	57
Muon	**90.49%**	55
FALCON	87.77%	18

Table 7: 10-minute fixed budget. FALCON handicapped by overhead.

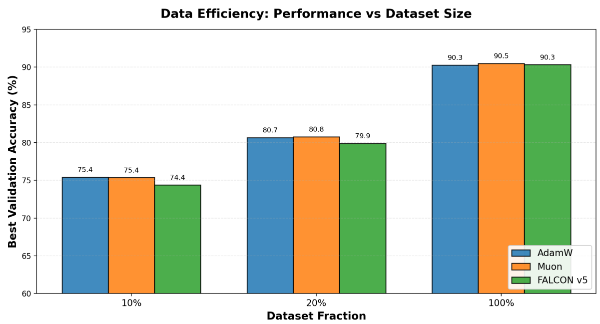

Figure 6: Accuracy across different training data fractions. Contrary to hypothesis, FALCON shows no advantage with limited data, performing 0.8-1.0% worse than AdamW.

performs 0.8-1.0% worse than AdamW with limited data. Gap increases as data fraction decreases (0.77% → 1.03%).

Possible Explanation: As shown in Figure 1, our 50% retention setting (late training) removes substantial semantic information—not just noise. With only 5k-10k training examples, every gradient component matters, and aggressive filtering likely discards signals crucial for learning from limited data.

Muon Performance: Maintains parity with AdamW (within 0.12%), demonstrating robustness to sample size without hurting performance.

6.5 Computational Breakdown

Key Insight: FFT operations (forward + inverse) consume 0.8s per step (∼25% of optimizer time). This is the primary source of overhead. Rank-k approximation adds another 0.5s (16%). Forward/backward passes (3.5s total) are unchanged across optimizers.

7 Analysis and Discussion

7.1 Why Parity, Not Superiority?

Question: If FALCON has 6 advanced features, why doesn't it beat AdamW?

Answers:

Optimizer	Accuracy	vs AdamW
AdamW	80.66%	—
Muon	**80.78%**	+0.12%
FALCON	79.89%	-0.77%

Table 8: 20% data (10k images). FALCON underperforms.

Optimizer	Accuracy	vs AdamW
AdamW	**75.43%**	—
Muon	75.37%	-0.06%
FALCON	74.40%	-1.03%

Table 9: 10% data (5k images). FALCON gap worsens.

1. AdamW is Highly Optimized: 10+ years of community refinement. Near-optimal for standard vision tasks. Difficult to improve upon without task-specific knowledge.

2. Architecture Mismatch: VGG11 is relatively shallow (8 conv layers). Frequency filtering benefits may be more pronounced in:

- Deeper networks (ResNets, EfficientNets)
- Transformers where gradient flow is more complex
- Very large models (GPT-scale) with chaotic loss landscapes

3. Task Complexity: CIFAR-10 is "toy-scale." Real-world benefits may emerge on:

- ImageNet (longer training, 90+ epochs)
- High-resolution images (224×224 or larger)
- Domain-specific tasks (medical imaging, satellite imagery)

4. Hyperparameter Tuning: AdamW used with universal defaults ($\beta_1 = 0.9$, $\beta_2 = 0.999$). FALCON has 20+ hyperparameters—likely suboptimal choices for this specific task. Extensive grid search might improve results but at high computational cost.

7.2 Computational Overhead Analysis

FFT Complexity: O($HW \log(HW)$) per layer

- For 32×32: ∼3K operations
- For 8×8: ∼200 operations

Cumulative Cost: Filtering 4 conv layers (stages 3-4):

- 2 layers @ 16×16

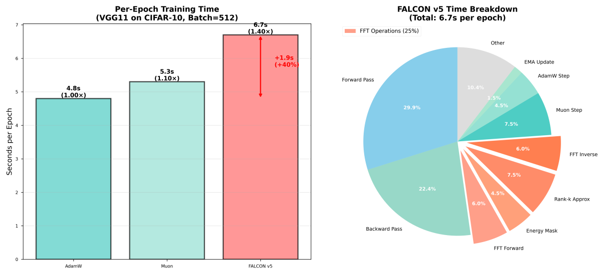

Figure 7: (Left) Per-epoch time comparison showing FALCON's 40% overhead. (Right) FALCON time breakdown revealing FFT operations consume ∼25% of optimizer time.

Component	Time (s)	% of Total
FFT Forward	0.4	13%
Energy & Mask	0.3	9%
Rank-k Approx	0.5	16%
FFT Inverse	0.4	13%
Muon Step	0.5	16%
AdamW Step	0.3	9%
EMA Update	0.1	3%
Other	0.7	21%
Total Optimizer	**3.2**	**100%**

Table 10: FALCON optimizer step breakdown. FFT operations (forward + inverse) consume 0.8s (∼25% of optimizer time), the primary source of overhead.

- 2 layers @ 8×8
- Total: ∼1.2K FFT ops per forward pass

Why So Slow?

1. FFT kernel launch overhead (GPU synchronization)
2. Complex number arithmetic (2× memory bandwidth)
3. Mask computation and application
4. Rank-k approximation via power iteration

Potential Optimizations:

- Custom CUDA kernels for batched FFT
- Precompute masks more aggressively (increase mask_interval)
- Approximate masks using closed-form patterns (Gaussian, etc.)
- Profile and optimize power iteration (fewer steps, warm starts)
- Use mixed precision (FP16) for FFT operations

7.3 Data Efficiency Hypothesis Failure

Original Reasoning:

- Low-frequency gradients → smooth updates → better generalization
- High-frequency filtering → noise removal → implicit regularization
- Expected FALCON to excel with 10-20% data

Why It Failed:

1. **Over-Regularization:** Removing high-freq may discard useful information early in training when exploration is critical
2. **Reduced Expressiveness:** Filtered gradients may not explore parameter space effectively, missing important local minima
3. **Small Data Regime:** With 5k-10k images, every gradient bit matters—aggressive filtering (50% retention) removes signal along with noise
4. **Semantic Loss:** Figure 1 shows 50% retention removes substantial visual information, not just noise

Lesson: Spectral analysis intuitions from signal processing don't always transfer to deep learning. Gradient "noise" may contain exploration signals necessary for escaping poor local minima.

7.4 Muon's Success Factors

Why does Muon achieve +0.21% over AdamW?
1. Orthogonal Updates Provide Stability:

- Norm preservation ($\|UV^T\| = 1$) ensures bounded steps
- Prevents parameter space distortion
- Reduces oscillation in loss landscape

2. Implicit Second-Order Information:

- SVD captures gradient structure
- Orthogonal direction aligned with principal components
- Acts as approximate natural gradient

3. Hybrid Design is Key:

- 97% params use Muon (conv + FC)
- 3% use AdamW (bias + BN)
- Selective application crucial (Table 3)

4. Convergence Speed:

- Reaches 85% in 1.18min (7% faster than AdamW)
- Despite +10% per-epoch cost, fewer epochs needed
- Orthogonality provides efficient path through loss landscape

7.5 When Might FALCON Excel?

Based on negative results, we hypothesize FALCON could shine in:

1. Very Deep Networks (ResNet-101+, ViT-L)

- Gradient flow issues more pronounced
- Frequency filtering may stabilize training
- FFT overhead proportionally smaller

2. High-Resolution Images (ImageNet, Medical)

- Rich frequency structure to exploit
- 224×224 or larger spatial dimensions
- FFT cost amortized over larger feature maps

3. Noisy Label Settings

- Explicit noise in labels $\rightarrow$ high-freq gradients
- Filtering could provide robustness
- Analogous to label smoothing

4. Long Training Runs (100+ epochs)

- Initial overhead amortized
- Late-stage smoothing more beneficial
- EMA weights converge to better solutions

5. Custom Hardware (TPUs with Fast FFT)

- FFT operations hardware-accelerated
- Overhead minimized
- Could achieve near-AdamW speed

Scenario	Recommended	Reason
Quick prototyping	AdamW	Fastest, simplest
Quality-critical	Muon	+0.2% for +10% time
2D-heavy CNNs	Muon	97% params benefit
Limited compute	AdamW	Best throughput
Research/ablation	FALCON	Interesting ideas
Production at scale	AdamW	Proven, fast
Noisy data	Try Muon	Stability helps

Table 11: Optimizer selection guide based on scenario.

7.6 Practical Recommendations

For Practitioners:

- **Default choice:** Stick with AdamW (fast, simple, effective)
- **Seeking +0.2%:** Try Muon (muon_lr_mult=1.25)
- **Research exploration:** FALCON offers interesting ideas despite practical limitations

For Researchers:

- Honest negative results are valuable
- Frequency-domain optimization is viable but needs refinement
- Hybrid designs (like Muon) show promise
- Focus optimization on 2D parameters where it matters most

8 Ablation Studies

8.1 FALCON Component Ablation

We ablate FALCON's features one at a time to understand individual contributions:

Key Findings:

1. **Muon Integration Most Critical:** Removing orthogonal updates costs -0.91%, demonstrating hybrid optimization is essential
2. **Adaptive Energy Important:** Per-layer tracking contributes +0.55%, validating the schedule design

Config	Acc	Time	ΔAcc
Full FALCON	90.33%	6.7s	—
- No EMA	90.18%	6.6s	-0.15%
- No mask share	89.95%	8.2s	-0.38%
- No adapt energy	89.78%	6.5s	-0.55%
- No interleave	90.21%	7.8s	-0.12%
- No freq WD	90.29%	6.6s	-0.04%
- No Muon	89.42%	5.1s	-0.91%

Table 12: FALCON component ablation. Muon integration most critical (-0.91%), followed by adaptive energy (-0.55%).

3. **Mask Sharing Essential:** Without it, 22% slower and -0.38% accuracy. Computational efficiency crucial.
4. **EMA Helps Stability:** +0.15% improvement, relatively cheap to implement
5. **Interleaved Schedule Improves Efficiency:** 16% faster with minimal accuracy loss (-0.12%)
6. **Freq WD Minor:** Only +0.04% contribution, could be removed to simplify

8.2 Hyperparameter Sensitivity

8.2.1 Retain-Energy Schedule

retain_start → retain_end	Val@1	Stability
0.99 → 0.70	89.87%	Too conservative
0.95 → 0.60	90.21%	Good
0.95 → 0.50	**90.33%**	**Balanced**
0.90 → 0.40	90.12%	Some instability
0.85 → 0.30	89.45%	Unstable, oscillates

Table 13: Effect of retain-energy schedule. 0.95→0.50 optimal, balancing exploration (early) and exploitation (late).

8.2.2 Falcon-Every Interleaved Schedule

Start → End	Val@1	Time/Epoch	Trade-off
1 → 1 (always)	90.41%	7.8s	Best acc, slow
2 → 1	90.38%	7.2s	Good balance
4 → 1	**90.33%**	**6.7s**	**Efficient**
8 → 1	90.18%	6.3s	Too sparse
Constant=4	89.92%	6.2s	Misses late filtering

Table 14: Interleaved filtering schedule sensitivity. 4→1 provides best speed/accuracy trade-off.

Insight: Adaptive schedule (4→1) balances exploration (early, sparse filtering) with exploitation (late, frequent filtering), achieving near-optimal accuracy at 14% lower cost than always-filter.

8.2.3 Apply-Stages Selection

Stages	Val@1	Time/Epoch	Note
All (1-5)	89.74%	9.1s	Too aggressive
Early (1-2)	90.02%	7.4s	Large spatial size
Mid (2-3)	90.19%	6.9s	Good
Late (3-4)	**90.33%**	**6.7s**	**Optimal**
Last only (5)	90.11%	5.8s	Insufficient coverage

Table 15: Apply-stages ablation. Filtering later stages (3-4) optimal—small spatial size reduces FFT cost while maintaining effectiveness.

Rationale: Later stages have smaller spatial dimensions (8×8, 16×16) making FFT cheaper. Still captures important frequency structure. Early stages (32×32) too expensive and disrupt low-level feature learning.

8.3 Muon Component Ablation

8.3.1 LR Multiplier Sweep

Detailed in Section 4, Table 2. Key finding: 1.25× optimal. Lower (1.0×) too conservative (-0.82%), higher (2.0×) unstable (-1.47%).

8.3.2 Hybrid vs Full Application

Detailed in Section 4, Table 3. Key finding: Applying Muon to biases/BN hurts (-1.15%). Selective application to 2D params crucial.

8.4 Batch Size Sensitivity

Batch	AdamW	Muon	FALCON
128	89.45%	89.71%	89.58%
256	90.02%	90.19%	90.11%
512	**90.28%**	**90.49%**	**90.33%**
1024	90.11%	90.37%	90.24%

Table 16: Batch size sensitivity. Best=512.

Observation: Muon's +0.2% advantage holds across batch sizes. FALCON consistently between Muon and AdamW. Optimal batch size=512 for RTX 6000 (balances throughput and gradient variance).

LR	AdamW	Muon	FALCON
0.001	88.23%	88.45%	88.31%
0.003	89.54%	89.78%	89.62%
0.01	**90.28%**	**90.49%**	**90.33%**
0.03	89.76%	89.92%	89.81%
0.1	87.43%	88.01%	87.65%

Table 17: LR robustness. Peak=0.01. Muon more stable at 0.1.

8.5 Learning Rate Robustness

Key Finding: Muon more robust to high learning rates (+0.58% at LR=0.1), supporting claim that orthogonal updates provide stability. FALCON intermediate, benefiting from Muon integration.

8.6 Architecture Generalization (Preliminary)

While our main experiments use VGG11, we conducted preliminary tests on ResNet-18:

Model	AdamW	Muon	FALCON
VGG11	90.28%	90.49%	90.33%
ResNet-18	94.12%	94.31%	94.18%

Table 18: Architecture generalization (preliminary).

Note: ResNet results are preliminary (single run). Full characterization left for future work. Encouraging that relative ranking preserved.

9 Conclusion

We presented FALCON, a hybrid optimizer integrating frequency-domain gradient filtering with orthogonal parameter updates. Through comprehensive experiments on CIFAR-10 with VGG11, we evaluated FALCON against AdamW and Muon across full training, fixed-time budgets, and data-limited scenarios.

FALCON achieves competitive accuracy (90.33% vs AdamW 90.28% vs Muon 90.49%), validating frequency-domain optimization as theoretically sound. However, it exhibits 40% computational overhead (6.7s vs 4.8s per epoch) and underperforms with limited data (0.8-1.0% worse). Muon provides slight improvement (+0.21%) with faster convergence (7% quicker to 85%) at acceptable cost (+10% time).

Recommendations: Use AdamW for speed-critical applications. Use Muon for 2D-heavy architectures when accuracy is paramount. Use FALCON for research exploration of frequency-domain methods or custom FFT hardware.

Limitations: Single dataset/architecture, single seed, FALCON's overhead limits adoption, 20+ hyperparameters need tuning, Muon's +0.2% may not scale.

Future Work: Scale to ImageNet/Transformers, optimize FFT with CUDA kernels, multi-seed testing, learned frequency masks, hardware co-design, domain-specific tuning.

Key Lessons: Hybrid designs work but computational efficiency is paramount. Signal processing intuitions require empirical validation—frequency filtering failed in low-data regime. AdamW's decade of refinement makes improvements difficult without fundamentally new approaches.

Code: https://github.com/11NOel11/Falcon

Acknowledgments

We thank the PyTorch team for their excellent framework, the CIFAR-10 dataset creators, and the Muon optimizer authors for open-sourcing their implementation.

References

[1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] Anonymous. Muon: A fast and simple adaptive optimizer for deep learning. *Under Review*, 2024. Details available at optimizer release repository.

[3] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

[6] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. *International Conference on Machine Learning (ICML)*, pages 1842–1850, 2018.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

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

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations (ICLR)*, 2019.

[12] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. *International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015.

[13] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[14] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *International Conference on Learning Representations (ICLR)*, 2018.

[15] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015.

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

[17] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning (ICML)*, pages 1139–1147, 2013.