ONE SIZE DOES NOT FIT ALL: ARCHITECTURE-AWARE ADAPTIVE BATCH SCHEDULING WITH DEBA

François Belias [1] **Naser Ezzati-Jivan** [2] **Foutse Khomh** [1]

Preprint. Under review at MLSys 2025.

ABSTRACT

Adaptive batch size methods aim to accelerate neural network training, but existing approaches apply identical adaptation strategies across all architectures, assuming a one-size-fits-all solution. We introduce DEBA (Dynamic Efficient Batch Adaptation), an adaptive batch scheduler that monitors gradient variance, gradient norm variation and loss variation to guide batch size adaptations. Through systematic evaluation across six architectures (ResNet-18/50, DenseNet–121, EfficientNet–B0, MobileNet-V3, ViT-B16) on CIFAR-10 and CIFAR-100 with five random seeds per configuration, we show that architecture fundamentally determines adaptation efficacy. Our findings reveal that: (1) lightweight and medium-depth architectures (MobileNet-V3, DenseNet–121, EfficientNet–B0) achieve a 45-62% training speedup with simultaneous accuracy improvements of 1-7%; (2) shallow residual networks (ResNet-18) show consistent gains of +2.4 - 4.0% in accuracy, 36 - 43% in speedup, while deep residual networks (ResNet-50) exhibit high variance and occasional degradation; (3) already-stable architectures (ViT-B16) show minimal speedup (6%) despite maintaining accuracy, indicating that adaptation benefits vary with baseline optimization characteristics. We introduce a baseline characterization framework using gradient stability metrics (stability score, gradient norm variation) that predicts which architectures will benefit from adaptive scheduling. Our ablation studies reveal critical design choices often overlooked in prior work: sliding window statistics (vs. full history) and sufficient cooldown periods (5+ epochs) between adaptations are essential for success. This work challenges the prevailing assumption that adaptive methods generalize across architectures and provides the first systematic evidence that batch size adaptation requires an architecture-aware design.

1 INTRODUCTION

Deep learning has achieved remarkable success across numerous domains, yet training inefficiency remains a critical bottleneck. State-of-the-art models often require weeks of computation on hundreds of GPUs, creating concerns about productivity, cost, and environmental impact (He et al., 2021; Strubell et al., 2019). A key algorithmic limitation lies in fixed optimization hyperparameters that fail to adapt as learning dynamics evolve throughout training. Batch size plays a pivotal role in determining both convergence speed and generalization (He et al., 2021). While fixed schedules remain dominant (Lau et al., 2025), they ignore evolving gradient noise and loss landscape curvature. Prior work shows that increasing batch size over time can reduce gradient variance (Smith et al., 2020), accelerate convergence (Goyal et al., 2017), and improve hardware utilization (You et al., 2017). However, overly large batches degrade generalization, suggesting that effective training requires schedules that adapt to changing optimization conditions.

Recent methods like AdaScale (Johnson et al., 2020) and GradTailor (Xu et al., 2020) exploit gradient statistics as feedback signals for adaptation. Gradients convey fine-grained information about learning stability, enabling decisions beyond coarse metrics like validation loss. Yet a crucial assumption remains unexamined: that a single adaptive strategy generalizes across architectures. Most methods are evaluated on one or two families (typically ResNets or Transformers) and implicitly assume transferability. This overlooks fundamental architectural differences. Architectures vary in optimization landscape smoothness (Li et al., 2018). They exhibit distinct gradient flow patterns (Wu et al., 2020). They differ in hyperparameter sensitivity (Liao et al., 2022).

[1]Department of Computer Engineering and Software Engineering, Polytechnique Montréal, Montréal, QC, Canada [2]Department of Computer Science, Brock University, St. Catharines, ON, Canada. Correspondence to: François Belias <francois-philippe.ossim-belias@polymtl.ca>, Naser Ezzati-Jivan <nezzati-jivan@brocku.ca>, Foutse Khomh <foutse.khomh@polymtl.ca>.

Proceedings of the 8^{th} MLSys Conference, Santa Clara, CA, USA, 2025.

In this work, we challenge the architecture-agnostic assumption underlying adaptive batch size methods. We present **DEBA (Dynamic Efficient Batch Adaptation)**, an adaptive scheduler that adjusts batch size based on training dynamics rather than following a fixed or uniform policy. We systematically evaluate DEBA across six diverse architectures—convolutional networks (ResNets, DenseNet, EfficientNet, MobileNet) and Vision Transformer—on CIFAR-10/100 with a rigorous experimental design. This cross-architecture design enables us to ask three research questions: (1) whether architecture fundamentally determines the effectiveness of adaptive batch scheduling, (2) whether baseline stability profiles predict adaptive performance, and (3) whether implementation decisions such as cooldown periods or statistical windows generalize across models.

Our results reveal that architecture fundamentally governs adaptation efficacy. Lightweight and medium-depth models achieve substantial speedup (up to 62%) while improving accuracy (up to +7%). Shallow residual networks yield consistent though smaller gains. Deeper residual networks exhibit unstable behaviour with occasional accuracy degradation. Already-stable architectures like ViT-B16 show minimal improvement, confirming that adaptation benefits diminish when gradient dynamics are inherently smooth. We further demonstrate that an architecture's compatibility with DEBA can be predicted through lightweight baseline profiling. A single fixed-batch run provides stability metrics that forecast whether adaptive scheduling will help or harm, enabling informed decisions about when to apply adaptation. Finally, our ablations reveal that implementation details are critical to success. Sliding-window statistics (vs. full-history averages), cooldown periods of at least five epochs between adjustments, and architecture-specific threshold tuning proved essential across all architectures, preventing oscillations and instability.

In summary, our contributions are:

1. **A systematic cross-architecture study** demonstrating that architecture fundamentally determines adaptive batch size efficacy, challenging one-size-fits-all assumptions in prior work.
2. **DEBA: A multi-signal adaptive scheduler** achieving simultaneous speedup (37-62%) and accuracy gains (up to +7%) on CIFAR datasets when configured adequately for target architectures.
3. **Predictive characterization framework** enabling a priori compatibility prediction through lightweight baseline profiling before expensive training.
4. **Critical design principles** identifying essential implementation choices: sliding window statistics, sufficient stabilization periods (5+ epochs), and architecture-specific threshold tuning.

Our findings suggest that adaptive batch-size methods benefit from architecture-specific tuning and baseline profiling rather than assuming universal applicability. They also indicate that future evaluations of adaptive techniques should include diverse architectures beyond the commonly used ResNets or Transformers. By bridging optimization dynamics with architectural characteristics, this work provides evidence for architecture-aware adaptive training. Code and experimental logs will be released in our replication package (Anonymous, 2025).

2 Related Works

Batch size is a critical hyperparameter that governs the trade-off between convergence speed, gradient noise, and generalization in stochastic optimization. Small batches introduce noise that helps escape sharp minima and improve generalization (Keskar et al., 2017), while large batches reduce variance and enable efficient parallelism (Goyal et al., 2017). This fundamental tension has motivated extensive research on batch size scheduling, yet the role of architecture in determining adaptation effectiveness remains largely unexplored.

2.1 Adaptive Batch size Scheduling

Dynamic batch size adjustment has emerged as a method to accelerate training while maintaining generalization. Early work established that increasing batch size can substitute for learning rate decay by reducing gradient noise (Smith et al., 2018). (Goyal et al., 2017) introduced the linear scaling rule with warmup schedules, enabling large-batch training for ResNet-50 on ImageNet without accuracy loss. (Shallue et al., 2019) studied the limits of batch-size scaling and found that architectures with different connectivity or normalization schemes exhibit varying tolerance to batch growth.

Recent methods propose adaptive schedules based on training dynamics. (You et al., 2020) introduced LAMB for layer-wise learning rate adaptation in large-batch settings. (Xu et al., 2020) adjusted the batch size on gradient cosine similarity in Transformers, exploiting the intuition that early training benefits from noisy gradients for exploration while later stages profit from larger batches for refinement. (McCandlish et al., 2018) investigated the relationship between gradient noise scale and optimal batch sizes, establishing theoretical bounds on efficient gains.

However, these works focus on demonstrating general applicability rather than systematically investigating how architectural characteristics determine adaptation effectiveness. While methods like LAMB perform layer-wise adaptation within architectures and validate across ResNets and Transformers, they treat architectural design as a given rather than

analyzing how global architectural properties, such as residual depth, connectivity patterns, and normalization schemes, influence the effectiveness of batch-size adaptation. No prior work compares how the same adaptive policy performs across diverse architectural families (residual, dense, efficient, attention-based designs) or provides a framework to predict which architectures will benefit from adaptive scheduling before expensive training runs.

2.2 Gradient-Based Training Monitoring

Gradient statistics provide fine-grained signals for training decisions. (Umeda & Iiduka, 2025) used gradient norm monitoring to detect training stagnation and adjust optimization strategies accordingly. (Xu et al., 2020) employed gradient cosine similarity as a gating mechanism for batch size adjustment, measuring alignment between consecutive updates to gauge convergence stability.

Other work has applied gradient monitoring to different optimization problems. (Liu et al., 2021) leveraged gradient variance estimation for adaptive learning rate schedules. (Vogels et al., 2020) used gradient structure analysis for communication-efficient distributed training. (Yu et al., 2020) exploited cosine similarity to resolve gradient conflicts in multi-task learning scenarios.

While these methods demonstrate the value of gradient-based monitoring, they rely on a single signal and do not account for the fundamental differences in gradient dynamics across architectures. Architectures with residual connections, for instance, maintain more stable gradient flow than vanilla networks, potentially altering the effectiveness of variance-based adaptation.

2.3 Architecture-Specific Optimization

The optimization dynamics of neural networks vary substantially with architectural design (Kaplan et al., 2020; Zhang et al., 2019). Residual connections improve gradient flow and tolerance to noise (He et al., 2021), dense connectivity smooths the loss landscape (Huang et al., 2018), and attention mechanisms alter curvature patterns (Xiong et al., 2020). Consequently, architectures exhibit different optimization behaviours under identical training protocols.

Recent work has begun to recognize these differences. (He et al., 2021) showed that ResNets and Transformers have different scaling properties in distributed settings. (You et al., 2017) introduced LARS specifically for ResNets with large batches, acknowledging that layer-wise adaptation needs vary across depth. Research on neural architecture search has revealed varying sensitivity to hyperparameters across designs (Zoph & Le, 2017; Real et al., 2019), suggesting that optimization strategies should account for architectural characteristics.

Studies on specific architectural patterns have documented their impact on training dynamics. Skip connections stabilize gradient magnitudes (He et al., 2021), batch normalization reduces internal covariate shift (Ioffe & Szegedy, 2015), and depthwise separable convolutions in efficient architectures create different parameter-gradient relationships (Howard et al., 2017). Despite this evidence, adaptive batch size methods treat architecture as a fixed background variable rather than a determining factor in adaptation effectiveness.

2.4 Our Approach

DEBA differs from prior adaptive batch scheduling methods in three key aspects. First, we conduct a systematic evaluation across six architectures, two datasets, and five random seeds, revealing clear architecture-dependent behaviour that challenges the assumption of universal applicability. Second, DEBA integrates three complementary gradient-based signals—variance, norm variation, and loss variation—combined with architecture-specific thresholds. This multi-signal design captures distinct aspects of training dynamics such as noise level, stability, and convergence behaviour, whereas no single metric proves sufficient across architectures. Third, through extensive ablation experiments, we identify critical design principles that govern adaptive scheduler robustness: sliding-window statistics for reliable estimation, stabilization periods of at least five epochs to prevent premature scaling, and architecture-aware threshold calibration. Together, these findings establish that adaptive batch sizing is inherently architecture-dependent and provide practical guidelines for assessing compatibility through lightweight baseline profiling.

3 THE DEBA ALGORITHM

This section provides a complete specification of the DEBA scheduler, addressing the algorithmic details necessary for reproducibility.

3.1 Overview and Design Philosophy

DEBA operates on the principle that batch size should adapt to the current optimization state rather than follow a fixed schedule. Unlike prior work that relies on single signals (e.g., gradient cosine similarity (Xu et al., 2020) or gradient noise scale (McCandlish et al., 2018)), DEBA combines three complementary signals that capture different aspects of training dynamics:

- **Gradient variance** (σ_g^2): Measures stochastic gradient noise, indicating confidence in the current update direction.
- **Gradient norm variation** ($\Delta\|\nabla L\|$): Tracks magnitude changes in gradients across iterations, signaling

optimization stability.

- **Loss variation** (ΔL): Monitors objective function changes, detecting convergence patterns or instability.

By jointly monitoring these signals, DEBA distinguishes between beneficial exploration noise (which should be preserved with smaller batches) and unnecessary variance (which can be reduced with larger batches). The scheduler incorporates a cooldown mechanism to prevent premature adaptations and uses architecture-specific thresholds derived from baseline profiling.

3.2 Signal Computation

At each epoch t, DEBA computes three signals from the training state. Let L_t denote the training loss and ∇_t the flattened gradient vector at epoch t.

Signal 1: Gradient Variance. This measures the spread of gradient components, reflecting SGD confidence:

$$\sigma^2_{g,t} = \text{Var}(\nabla_t) = \frac{1}{|\nabla_t|}\sum_{i=1}^{|\nabla_t|}(\nabla_t^{(i)} - \bar{\nabla}_t)^2 \tag{1}$$

where $\nabla_t^{(i)}$ is the i-th component of the flattened gradient and $\bar{\nabla}_t$ is the mean. High variance indicates noisy gradients beneficial for exploration; low variance suggests stable convergence suitable for larger batches.

Signal 2: Gradient Norm Variation. This tracks relative changes in gradient magnitude:

$$\Delta\|\nabla L\|_t = \frac{|\|\nabla_t\| - \|\nabla_{t-1}\||}{|\|\nabla_{t-1}\|| + \epsilon} \tag{2}$$

where $\epsilon = 10^{-8}$ prevents division by zero. Large variations indicate unstable optimization, potentially from sharp minima or batch size incompatibility.

Signal 3: Loss Variation. This measures objective function stability:

$$\Delta L_t = \frac{|L_t - L_{t-1}|}{|L_{t-1}| + \epsilon} \tag{3}$$

Rapid loss fluctuations suggest the current batch size may be inappropriate for the loss landscape geometry.

All three signals are tracked in a sliding window of 15 epochs to capture recent dynamics while discarding stale information from earlier training phases.

3.3 Decision Logic

DEBA employs a rule-based decision system that interprets the three signals to classify the current optimization state. The decision at epoch t depends on two normalized metrics:

Confidence Score. This quantifies gradient stability relative to recent history:

$$c_t = \frac{\sigma^2_{g,t}}{\text{median}(\{\sigma^2_{g,t-14}, \ldots, \sigma^2_{g,t}\}) + \epsilon} \tag{4}$$

A confidence score $c_t > \theta_{\text{conf}}$ indicates gradients are noisier than typical, suggesting the batch size should remain small or decrease.

Stability Indicators. Two binary flags assess optimization stability:

$$\text{stable\_gradients} = \mathbb{I}[\Delta\|\nabla L\|_t < \theta_{\text{stab}}] \tag{5}$$

$$\text{stable\_loss} = \mathbb{I}[\Delta L_t < \theta_{\text{stab}}] \tag{6}$$

where $\mathbb{I}[\cdot]$ is the indicator function, θ_{conf} is the confidence threshold, and θ_{stab} is the stability threshold.

The decision rule combines these indicators:

$$\text{decision}_t = \begin{cases} \text{increase}, & \text{if } c_t \le \theta_{\text{conf}} \wedge \text{stable\_gradients} \\ & \wedge\ \text{stable\_loss} \\ \text{rollback}, & \text{if } c_t > \theta_{\text{conf}} \vee \Delta\|\nabla L\|_t > 3\theta_{\text{stab}} \\ \text{hold}, & \text{otherwise.} \end{cases} \tag{7}$$

This logic encodes three behavioural patterns:

- **Increase**: Low gradient variance with stable norms and loss indicates the model can handle larger batches, improving throughput without sacrificing exploration.
- **Rollback**: High variance or severe gradient instability suggests the current batch size exceeds the architecture's tolerance, risking convergence to sharp minima. The batch size is reduced to restore beneficial noise.
- **Hold**: Ambiguous signals warrant maintaining the current configuration until clearer evidence emerges.

3.4 Batch Size Update Mechanism

Based on the decision decision_t, DEBA updates the batch size according to:

$$B_{t+1} = \begin{cases} \min(B_{\max}, \lfloor \alpha_{\text{grow}} \cdot B_t \rfloor) & \text{if decision}_t = \text{increase} \\ \max(B_{\min}, \lfloor \alpha_{\text{roll}} \cdot B_t \rfloor) & \text{if decision}_t = \text{rollback} \\ B_t & \text{if decision}_t = \text{hold} \end{cases} \tag{8}$$

where B_t is the batch size at epoch t, $\alpha_{\text{grow}} = 1.5$ is the growth factor, $\alpha_{\text{roll}} = 0.8$ is the rollback factor, $B_{\min} = 16$, and $B_{\max} = 2048$. These bounds prevent degenerate cases (tiny batches causing slow training or memory overflow due to excessive growth).

3.5 Cooldown Period

To prevent oscillations and allow the model to stabilize after batch size changes, DEBA enforces a cooldown period T_{cool} between consecutive adaptations. After any increase or rollback at epoch t_{last}, no further adaptations occur until epoch $t > t_{\text{last}} + T_{\text{cool}}$. During cooldown, signals are still computed and logged, but decisions default to "hold."

Empirically, we find $T_{\text{cool}} = 5$ epochs provides sufficient time for BatchNormalization statistics and optimizer momentum to recalibrate (Section 6). Shorter cooldowns ($T_{\text{cool}} = 2$) cause "decision thrashing" with accuracy losses of 2.3–12.6 points, while longer cooldowns ($T_{\text{cool}} = 10$) improve accuracy at modest speedup cost.

3.6 Architecture-Specific Threshold Calibration

The thresholds θ_{conf} and θ_{stab} are architecture-dependent, reflecting fundamental differences in gradient scale and loss landscape properties. Table 1 presents the calibrated values used in our experiments. These thresholds are obtained through the baseline profiling procedure described in Section 5. A fixed-batch run measures natural gradient statistics, and thresholds are set at the 75th percentile of observed gradient-norm variation and at the median of gradient-variance ratios. This calibration is computationally inexpensive (a single training run) and substantially outperforms universal thresholds, which degrade performance across 3/6 architectures by 1.7–3.3 points (Section 6).

3.7 Computational Overhead

DEBA's per-epoch overhead consists of three operations: (1) computing gradient variance via a single pass over the flattened gradient vector ($\mathcal{O}(P)$ where P is parameter count), (2) computing gradient norm ($\mathcal{O}(P)$), and (3) maintaining sliding window statistics ($\mathcal{O}(1)$ deque operations). The total overhead is $\mathcal{O}(P)$ per epoch, which is negligible compared to the $\mathcal{O}(P \cdot N \cdot E)$ cost of forward-backward passes over N samples for E epochs. In practice, signal computation adds $< 1\%$ to wall-clock training time on all tested architectures.

4 RQ_1: Does architecture fundamentally determine the effectiveness of adaptive batch size methods?

4.1 Motivation and Hypothesis

Conventional wisdom suggests a universal trade-off between batch size, training speed, and generalization. Larger batches improve throughput by leveraging parallelism, yet they reduce stochastic gradient noise, which can limit exploration and lead to poorer generalization (Keskar et al., 2017; Hoffer et al., 2018; Smith et al., 2020). However, recent evidence indicates that this trade-off is not universal but instead architecture-dependent. Deep residual and transformer architectures are typically more sensitive to reductions in gradient noise due to their highly non-linear composition, which increases the likelihood of converging to sharp minima (Keskar et al., 2017; Goyal et al., 2017). Conversely, lightweight or densely connected convolutional networks such as MobileNet, EfficientNet, and DenseNet tend to exhibit smoother loss landscapes and greater tolerance to large batch sizes (You et al., 2017; Shallue et al., 2019). These observations suggest that adaptive batch size strategies may yield architecture-specific benefits.

We hypothesize that the effectiveness of adaptive batch size methods such as DEBA is governed primarily by each architecture's intrinsic gradient stability and its sensitivity to variance reduction. Architectures characterized by smoother loss surfaces and higher noise tolerance should benefit more consistently from batch size adaptation, realizing both faster training and better generalization. In contrast, deeper or more sensitive architectures may experience diminishing returns or even degradation when batch size grows too aggressively.

4.2 Experimental Setup

We evaluate DEBA against a fixed batch size regime using five random seeds (2, 7, 42, 123, 199) across CIFAR-10 and CIFAR-100. Six architectures: ResNet-18, ResNet-50, MobileNet-V3, DenseNet–121, EfficientNet–B0, and ViT-B16 are chosen to cover a broad spectrum of depth, parameterization, and stability characteristics. All models are trained for 100 epochs with SGD (momentum 0.9, weight decay 5×10^{-4}) and an initial batch size of 64. Under DEBA, batch size is adapted every 10 epochs according to training stability metrics (see Section 6). All experiments are conducted on a single NVIDIA H100 GPU, and we report wall-clock time and final test accuracy.

4.3 Results: Architecture-Dependent Patterns

Table 4 summarizes results across all twelve configuration pairs. The outcomes reveal clear architecture-dependent trends that directly address our RQ_1. The speedup ranges from 8% to 62%, while accuracy variations span from a 0.6% drop to a 7.0% improvement.

Three consistent behavioural groups emerge. High-benefit architectures such as MobileNet-V3, DenseNet–121, and EfficientNet–B0 achieve between 45–62% faster training while simultaneously improving accuracy by 0.4–7.0%. These gains are stable across both datasets, with MobileNet-V3 showing a particularly pronounced improvement on CIFAR-10 (+7.0% accuracy) and DenseNet–121 achieving a 3.2% gain on CIFAR-100 with 62.4% speedup. ResNet-18

shows moderate but reliable benefits, improving by 2–4% in accuracy and 36–43% in speed, likely because its relatively shallow depth enables stable adaptation without over-regularization. In contrast, ResNet-50 and ViT-B16 exhibit limited or unstable benefits. ViT-B16 achieves only marginal speedup (5–8%) with unchanged accuracy, while ResNet-50, despite a 43% speedup, shows a 0.6% accuracy drop on CIFAR-10 and high variance across seeds on CIFAR-100, indicating sensitivity to initialization and training dynamics.

4.4 Mechanistic Interpretation

The speedup mechanism is straightforward: as illustrated in Figure 1, time per epoch decreases exponentially with batch size due to improved GPU utilization. DEBA exploits this by starting with small batches to preserve exploratory noise, then gradually increasing the batch size to leverage hardware parallelism, achieving 30–50% reductions in wall-clock time.

Accuracy improvements stem from gradient dynamics. As shown in Figure 2, gradient variance exhibits a U-shaped relation with batch size, where moderate ranges (BS=128–512) balance stability and stochastic regularization. In high-benefit architectures (ResNet-18, DenseNet-121, EfficientNet-B0, MobileNet-V3), DEBA reduces gradient variance by 65–85%, achieving smoother convergence while maintaining beneficial stochasticity through periodic rollback corrections when variance becomes excessively low.

ResNet-50 exposes the principal failure mode: excessive gradient stabilization (65% variance reduction) suppresses stochasticity required for generalization, leading to sharper minima (stability score: 0.203→0.159, gradient norm oscillations: 2.4× increase). This produces highly seed-dependent outcomes ranging from +2.17% to -3.89% accuracy, confirming that deeper architectures amplify noise imbalances from aggressive batch scaling.

The temporal evolution follows three phases: early epochs (1–30) preserve high variance (10^{-5}–10^{-6}) with small batches (64–216) for exploration; mid-training (30–70) stabilizes convergence with gradual increases (216–729); late stages (70–100) use large batches ($\geq$1024) for acceleration, with occasional rollbacks to sustain generalization. These dynamics enable 36–62% speedup with accuracy gains in 9/12 cases, while revealing architectural boundaries for excessively deep or variance-sensitive models.

Summary: Overall, RQ_1 demonstrates that the efficacy of adaptive batch size scheduling is not universal but fundamentally shaped by architectural characteristics. The interplay between loss landscape smoothness, gradient variance tolerance, and exploration-exploitation balance dictates whether DEBA yields synergy or instability.

Table 1. Architecture-specific thresholds derived from baseline profiling.

Architecture	θ_{stab}	θ_{conf}
MobileNet-V3	0.25	0.5
EfficientNet–B0	0.60	0.8
ResNet-18	0.55	0.6
ResNet-50	12.0	0.8
DenseNet–121	0.55	0.6
ViT-B16	0.40	0.7

5 RQ_2: Can gradient stability metrics predict architecture compatibility with adaptive batch scheduling?

5.1 Motivation and Hypothesis

The success of adaptive optimization techniques often depends on their compatibility with architectural characteristics that shape optimization dynamics. Prior work has shown that neural network architectures differ markedly in their loss landscape geometry (Li et al., 2018), gradient flow properties (Wu et al., 2020), and sensitivity to hyperparameter choices (Liao et al., 2022). Consequently, hyperparameter configurations that perform well for one architecture, such as ResNet, can fail dramatically for another, such as MobileNet. This variability complicates both tuning and the deployment of adaptive methods, raising an important question: can we predict in advance whether an architecture will respond positively to adaptive batch scheduling?

We hypothesize that architectural compatibility with adaptive batch size methods such as DEBA is not arbitrary, but instead emerges from the model's intrinsic training stability under standard (fixed-batch) conditions. Specifically, we posit that an architecture's natural stability profile captured through gradient and loss dynamics across multiple random seeds acts as a reliable predictor of its adaptive behaviour. Architectures with moderate stability are expected to benefit most from DEBA, balancing responsiveness to adaptation with robustness to perturbations. Overly stable architectures may profit from DEBA's controlled disruption, which injects beneficial noise to escape flat or stagnant regions. Conversely, inherently unstable architectures are expected to show seed-dependent or inconsistent results, as DEBA cannot compensate for fundamental gradient instability.

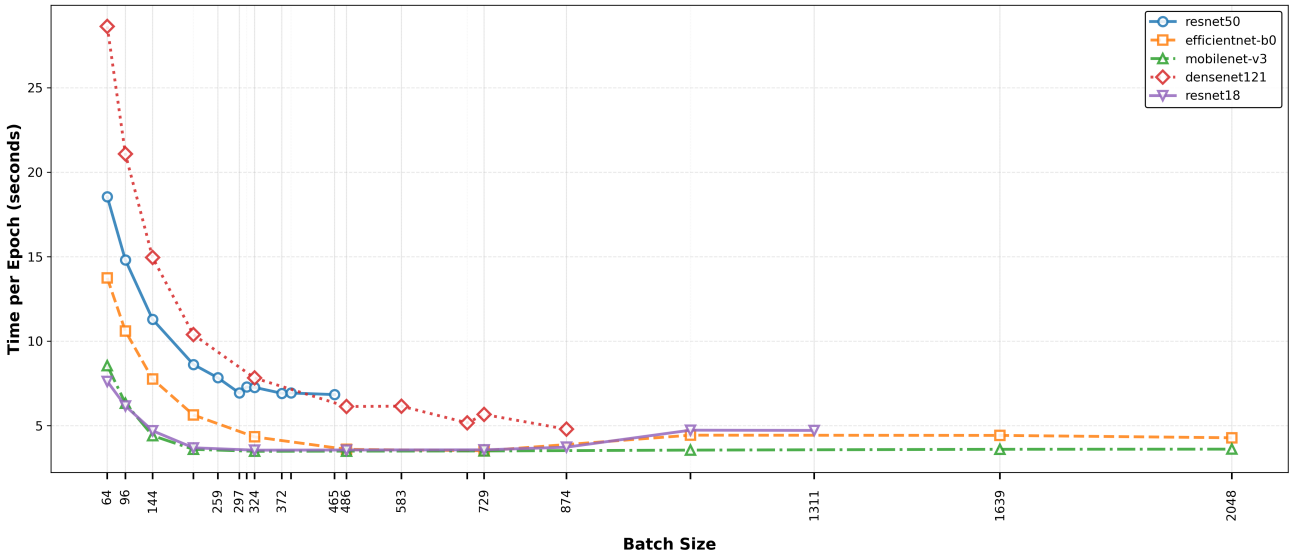

Figure 1. Impact of batch size on training speed.

5.2 Methodology: Stability Profiling Framework

To operationalize this hypothesis, we introduce a quantitative stability profiling procedure based on fixed-batch training. For each architecture, we measure per-epoch gradient variance σ_g^2, variation in gradient norm $\text{Var}(\|\nabla\mathcal{L}\|)$, and variation in loss $\text{Var}(\mathcal{L})$. These quantities are aggregated into a composite stability score S, defined as:

$$S = \frac{1}{1 + \text{CV}(\sigma_g^2) + \overline{\text{Var}(\|\nabla\mathcal{L}\|)} + \overline{\text{Var}(\mathcal{L})}}, \tag{9}$$

where CV denotes the coefficient of variation and $\bar{\cdot}$ indicates mean values across epochs. By construction, higher S values correspond to smoother, more stable optimization trajectories, while lower scores reflect volatile or chaotic training behaviour.

Each architecture is trained for 100 epochs under two conditions: (1) a fixed batch size of 64, establishing a baseline stability profile, and (2) the DEBA regime, which adapts batch size dynamically according to gradient statistics. We run three random seeds (2, 42, 199) to estimate both the mean stability μ_S and seed sensitivity σ_S. DEBA-specific metrics such as decision aggressiveness, i.e the rate of batch size change, and convergence stability (standard deviation of loss over time) are also recorded. Architectures are categorized by their baseline μ_S values and analyzed for consistency between fixed-batch stability and DEBA performance.

5.3 Results and Analysis

Table 2 summarizes the profiling outcomes, revealing four distinct stability archetypes that correspond closely to DEBA compatibility patterns.

Overly Stable Architectures. MobileNet-V3 and ViT-B-16 achieve high baseline stability ($\mu_S > 0.54$, $\sigma_S < 0.03$), yet their adaptive responses diverge sharply. MobileNet-V3 gains +5.3% accuracy as DEBA's negative decision aggressiveness (-0.39) triggers frequent rollbacks, intentionally destabilizing the model (μ_S: 0.590→0.258) to escape suboptimal minima. Conversely, ViT-B-16's extremely low gradient variance ($\sigma_g^2 \approx 5 \times 10^{-7}$) and smooth attention-based landscape leave minimal room for improvement (+0.2%), as its near-deterministic optimization path remains largely unaffected by batch size changes.

Moderately Stable Architectures. ResNet-18 and DenseNet-121 fall within the optimal stability range ($0.48 \leq \mu_S \leq 0.50$), achieving consistent gains (2–3% accuracy, 36–43% speedup). DEBA reduces gradient variance by 69–85% and gradient norm variation by 73%, smoothing convergence without eliminating stochasticity. Their high decision aggressiveness (0.52–0.7) demonstrates tolerance to rapid batch scaling, facilitated by residual/dense connections that preserve gradient flow. These results validate the "sweet spot" hypothesis, in which DEBA reinforces convergence efficiency without destabilization.

Dynamically Stable Architecture EfficientNet-B0 exhibits intermediate stability ($\mu_S = 0.331$) but extreme seed sensitivity ($\sigma_S = 0.182$). DEBA consistently improves accuracy by 1.5–2.4% through responsive rollback mechanisms (aggressiveness -0.02 to -0.12) that counteract volatility, demonstrating intelligent navigation of chaotic dynamics rather than pure stabilization.

Naturally Unstable Architecture ResNet-50 ($\mu_S = 0.185$,

$\sigma_S = 0.079$) exhibits highly seed-dependent outcomes: favourable initializations yield +2.17% accuracy, while unstable ones cause -1.6% degradation. Its depth amplifies gradient interactions, causing DEBA's aggressiveness (0.28–0.4) to exacerbate rather than correct instability, confirming that naturally unstable architectures are unreliable candidates for adaptive scheduling.

5.3.1 Predictive Framework

Baseline stability scores reliably predict DEBA outcomes: architectures within $\mu_S \in [0.45, 0.55]$ consistently benefit; those above 0.55 improve only with convergence stability $<$ 0.20 (indicating stagnation); models below 0.26 show seed-dependent behaviour. EfficientNet-B0 is correctly identified as "dynamically stable" via $\sigma_S > 0.15$. Decision aggressiveness metrics further confirm DEBA's implicit architecture detection: stable models tolerate rapid scaling (0.52–0.7), while sensitive ones require caution (0.28–0.45), with negative values (-0.39 to -0.10) indicating active rollback cycles.

Summary: RQ_2 demonstrates that an architecture's intrinsic training stability provides a strong prior for predicting its response to adaptive batch size scheduling. The stability score S computed from fixed-batch runs captures the essential dynamics that govern DEBA's effectiveness: moderate-stability architectures benefit consistently, overly stable ones gain from controlled disruption, dynamically stable ones profit from adaptive correction, and naturally unstable ones remain unreliable. This predictive framework offers an alternative to trial-and-error adaptive training, enabling architecture-aware deployment of batch size adaptation strategies.

Table 2. Architecture Stability Taxonomy (Fixed BS Profiling)

Architecture	μ_S	σ_S	Class	ΔAcc
MobileNet-V3	0.590	0.028	Overly Stable	+5.3%
ViT-B-16	0.546	0.012	Overly Stable	+0.2%
DenseNet-121	0.495	0.019	Moderately Stable	+1.9%
ResNet-18	0.487	0.015	Moderately Stable	+2.6%
EfficientNet-B0	0.331	0.182	Dynamically Stable	+1.9%
ResNet-50	0.185	0.079	Naturally Unstable	-0.3%

6 RQ_3: What implementation choices critically affect adaptive scheduler robustness across architectures?

6.1 Motivation

Section 4 demonstrates that architecture fundamentally determines adaptive batch size effectiveness, with speedup varying from 8% to 62% and accuracy changes from -0.6% to +7.0%. We investigate whether these outcomes reflect intrinsic architectural properties or scheduler implementation artifacts through systematic ablations of: (1) threshold tuning strategy, (2) stabilization period, and (3) observation window for gradient statistics.

6.2 Methodology

Our baseline DEBA uses architecture-specific thresholds from profiling (Table 1), cooldown=10 epochs, sliding window=15 epochs, and growth/rollback factors of 1.5×/0.8×, achieving 36-62% speedup with accuracy gains on 5/6 architectures. All experiments use CIFAR-10, 100 epochs, 5 random seeds, with fixed BS=64 as comparison.

6.3 Results

Table 3 reveals three patterns: (1) threshold calibration and cooldown critically affect all architectures, (2) observation window has moderate, architecture-dependent effects, and (3) no single configuration succeeds universally.

6.3.1 Ablation 1: Threshold Calibration Strategy

Our baseline derives architecture-specific thresholds through profiling (Table 1), revealing a 48× range: θ_{stab} spans 0.25 (MobileNet-V3) to 12.0 (ResNet-50). Universal thresholds ($\theta_{\text{stab}} = 0.1$, $\theta_{\text{conf}} = 0.5$) catastrophically degrade half the architectures: ResNet-18 loses all benefits (ΔAcc: +2.43→-0.16, speedup: -10.8%), while DenseNet–121 and EfficientNet–B0 drop 1.7-3.3 points with speedup regressions to -25.5%. The failure reflects gradient scale mismatch—architectures with moderate gradient norm variation (0.4-0.6) find $\theta_{\text{stab}} = 0.1$ too restrictive, triggering excessive rollbacks. Conversely, MobileNet-V3's success (+7.02%) is coincidental, as its gradient scale (0.2-0.3) happens to match defaults. The 48× threshold range reflects fundamental architectural differences in gradient behaviour, not tuning artifacts.

6.3.2 Ablation 2: Stabilization Period Between Adaptations

Cooldown=2 universally degrades performance, with 5/6 architectures losing > 1 accuracy point. ResNet-50 suffers catastrophic collapse (-12.62 points despite 58.7% speedup), while even high-benefit MobileNet-V3 retains only marginal gains (+0.64 vs. +6.98 baseline). Training curves reveal "decision thrashing" where batch size oscillates every 2-3 epochs, preventing convergence.

Cooldown=5 recovers most architectures (+0.24 to +5.24 points, 50-67% speedup), though ResNet-50 remains problematic (-2.35 points). Cooldown=10 provides incremental accuracy improvements (ResNet-18: +2.43 vs. +0.83; MobileNet-V3: +6.98 vs. +5.24) at modest speedup cost (DenseNet–121: 67%→55%), representing a fundamental

speed-accuracy tradeoff.

Three processes require stabilization: BatchNormalization recalibration, momentum buffer adjustment, and loss landscape navigation after effective learning rate changes. Cooldown=2 violates all requirements, causing "adaptation shock." Cooldowns ≥ 5 provide sufficient recovery, with 10 epochs offering additional stability margin.

6.3.3 Ablation 3: Temporal Scope of Gradient Statistics

Full history (vs. sliding window=15) produces heterogeneous outcomes. EfficientNet–B0 and DenseNet–121 experience severe speedup collapse (52%→7%, 67%→37%) as stale statistics cause overly conservative decisions. Conversely, ResNet-18 and ResNet-50 show slight accuracy improvements (+1.94, +0.14) with reduced speedup—full history's conservatism prevents harmful scaling but sacrifices efficiency. MobileNet-V3 and ViT-B16 remain unaffected due to stable, slowly-varying gradient distributions.

The failure stems from two mechanisms: (1) stale reference statistics inflate baselines, triggering false rollback alarms, and (2) adaptation lag where historical averages dilute recent signals from qualitatively different training phases. Sliding windows detect regime transitions within 15 epochs, enabling responsive adaptation.

6.4 Design Principles for Robust Adaptive Scheduling

Universal requirements. Sufficient stabilization period (cooldown $\geq$ 5 epochs) is architecture-invariant. Cooldown=2 causes 2.3-12.6 point losses across all models due to optimization fundamentals: BatchNorm statistics, momentum buffers, and loss landscape trajectories require 3-5 epochs to equilibrate. We recommend a minimum cooldown of 5, increasing to 10 for high-seed-sensitivity models.

Sliding windows provide substantially more consistent behaviour than full history, balancing responsiveness with statistical stability. Full history causes unpredictable outcomes: speedup variance explosion (EfficientNet–B0: std 6%→53%) and average speedup collapse (DenseNet-: 67%→37%).

Architecture-specific requirements. Threshold calibration is non-negotiable. Universal thresholds degrade 50% of architectures (3/6) by 1.7-3.3 points with speedup regressions to -25.5%. The 48× variation in optimal θ_{stab} reflects fundamental gradient scale differences. We recommend baseline profiling (Section 5)—a single 100-epoch fixed-batch run —to derive architecture-specific thresholds.

Summary: RQ_3 reveals that implementation details are not created equal. Cooldown period is a hard constraint requiring minimum values regardless of architecture, while threshold calibration is essential but architecture-dependent. Combined with baseline profiling (Section 5), these principles provide a structured deployment procedure grounded in measurable gradient behaviour, reducing brittleness in adaptive methods.

7 Discussion

7.1 Key Insights and Generalization

Our findings show that the effectiveness of adaptive batch scheduling is not universal but fundamentally determined by architectural characteristics. Lightweight and densely connected models consistently benefit from DEBA, whereas very deep or inherently stable ones, such as ResNet-50 and ViT-B16, exhibit limited or unstable gains. These trends remain consistent across datasets and random seeds, indicating that the determining factor is architectural design rather than random variation. Although our experiments focus on CIFAR-scale data, the mechanisms identified—gradient variance control, exploration–stability balance, and architecture-dependent noise tolerance—are fundamental to stochastic optimization and are expected to generalize to larger-scale training scenarios.

7.2 Practical Implications

DEBA's profiling-based configuration provides a practical route toward architecture-aware adaptation. A fixed-batch profiling run can estimate gradient stability and guide the selection of architecture-specific thresholds, while a moderate cooldown between adaptations helps ensure stability. This lightweight procedure reliably reproduces DEBA's benefits and mitigates instability across architectures without the need for costly hyperparameter searches.

7.3 Limitations and Interactions

The proposed taxonomy, which distinguishes high-, moderate-, and low-benefit architectures, is empirical and may not directly apply to extremely deep or hybrid designs such as ConvNeXt or CoAtNet. Furthermore, our controlled setup isolates batch-size adaptation while keeping the learning rate fixed. In contrast, in realistic training pipelines, DEBA would interact with adaptive learning rate schedules, warmup phases, and normalization layers. Understanding these interactions and extending DEBA to co-adapt batch size and learning rate, as in AdaScale-like frameworks, represents an essential next step.

Table 3. Impact of implementation choices on DEBA performance (CIFAR-10, 5 seeds). Baseline uses architecture-specific thresholds, cooldown=10, and sliding window=15. ΔAcc shows accuracy change vs. fixed BS=64 baseline.

	Baseline		**Universal Thresh.**		**Cooldown=2**		**Cooldown=5**		**Full History**	
Architecture	ΔAcc	Speedup (%)	ΔAcc	Speedup (%)	ΔAcc	Speedup (%)	ΔAcc	Speedup (%)	ΔAcc	Speedup (%)
ResNet-18	+2.43	34.2	-0.16	-10.8	-2.75	40.7	+0.83	21.6	+1.94	31.8
ResNet-50	-0.59	42.4	+0.43	-5.6	-12.62	58.7	-2.35	43.4	+0.14	11.9
DenseNet-121	+1.40	54.9	-0.26	-0.05	-4.30	79.0	+0.24	67.0	+1.02	37.1
EfficientNet-B0	+2.37	43.3	-0.95	-25.5	-2.29	65.0	+1.97	52.0	+0.24	7.1
MobileNet-V3	+6.98	50.4	+7.02	37.9	+0.64	55.4	+5.24	50.2	+5.48	42.0
ViT-B16	-0.61	8.3	-0.02	-1.9	-1.80	11.9	-0.59	10.0	-0.43	6.8
Degraded (>1pt)	0/6	—	3/6	—	5/6	—	1/6	—	0/6	—
Mean $\lvert\Delta Acc$—	2.40	—	1.47	—	4.07	—	1.20	—	1.54	—

7.4 Future Directions

Future work will extend the evaluation of DEBA to larger-scale datasets such as ImageNet and to distributed training environments where profiling overhead becomes more pronounced. We also plan to explore meta-learning or transfer-based approaches that can infer optimal adaptation thresholds directly from architectural features, reducing manual calibration. Applying this framework to other domains, including natural language processing and reinforcement learning, would further test whether architecture-dependent stability generalizes across modalities. Finally, establishing formal connections between gradient statistics, Lipschitz smoothness, and adaptive behavior remains an important direction toward a theoretical foundation for architecture-aware optimization.

8 Threats to Validity

8.1 Internal validity

We fix the learning rate (0.01) and optimizer (SGD, momentum=0.9) to isolate batch size effects. While large-batch training often uses LR scaling (Goyal et al., 2017) or adaptive optimizers, our controlled setup ensures observed architecture-dependent patterns reflect intrinsic optimization characteristics rather than optimizer interactions. DEBA's hyperparameters (thresholds, cooldown) are tuned via baseline profiling (Section 5). Our ablations (Section 6) vary factors independently; subtle interactions may exist, but the 48× threshold range confirms architecture-agnostic tuning is fundamentally inadequate.

8.2 External validity

We evaluate on CIFAR-10/100 (32×32, 50k samples), enabling rigorous multi-seed evaluation but potentially understating large-scale dynamics (ImageNet: 1.2M samples, 224×224). Architecture-dependent trends remain consistent across both datasets (10 vs 100 classes), suggesting these are not dataset artifacts. Our six architectures span diverse paradigms (residual, dense, efficient, and attention-based) but do not cover all modern families (such as ConvNeXt and Swin). The baseline profiling method (Section 5) enables assessing new architectures without complete retraining. Single-GPU experiments (H100) offer maximal control; distributed settings may alter speedups due to communication overhead, though per-epoch time reductions stem from hardware-agnostic parallelism principles.

8.3 Statistical validity

We use five random seeds per configuration (60 runs total), exceeding norms in adaptive batch studies (Devarakonda et al., 2017; Smith et al., 2018). Our strongest claims rely on consistent, low-variance gains (std $<1.5\%$); high variance in problematic models (ResNet-50) is treated as an empirical finding. Despite fixed seeds, GPU-level randomness (cuDNN) may cause minor deviations (Pineau et al., 2020). Consistent trends across diverse seeds (2, 7, 42, 123, 199) demonstrate robustness.

9 Conclusion

This work challenges the assumption that a single adaptive batch size strategy can generalize across all neural network architectures. We introduced **DEBA**, a multi-signal scheduler that dynamically adjusts batch size based on gradient and loss dynamics. Through systematic evaluation across six architectures and two datasets, we showed that architecture fundamentally determines the effectiveness of adaptive batch scheduling. Lightweight and moderately deep networks consistently benefit, while very deep or intrinsically stable ones show limited or even negative responses, revealing three distinct regimes of architectural behaviour.

Beyond these empirical findings, DEBA establishes a practical framework for architecture-aware adaptation. Its gradient-based signals offer interpretable criteria for batch size scaling, and its baseline profiling procedure allows practitioners to anticipate compatibility through short fixed-

batch runs. This approach replaces heuristic tuning with predictive insight, enabling selective and efficient deployment of adaptive methods.

More broadly, our results indicate that adaptation in deep learning cannot remain architecture-agnostic. As models continue to diversify—from compact edge networks to billion-parameter transformers—training strategies must evolve toward systems that recognize and respond to architectural properties. DEBA thus represents both a methodological advance and a conceptual step toward principled, architecture-aware optimization. All code and experimental logs will be released in our replication package (Anonymous, 2025) to support reproducibility and future research.

REFERENCES

Anonymous. Replication package for the paper "one size does not fit all: Architecture-aware adaptive batch scheduling with deba", October 2025. URL https://doi.org/10.5281/zenodo.17478685.

Devarakonda, A., Naumov, M., and Garland, M. Adabatch: Adaptive batch sizes for training deep neural networks. *arXiv preprint arXiv:1712.02029*, 2017.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

He, X., Xue, F., Ren, X., and You, Y. Large-scale deep learning optimizations: A comprehensive survey, 2021. URL https://arxiv.org/abs/2111.00856.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks, 2018. URL https://arxiv.org/abs/1705.08741.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. URL https://arxiv.org/abs/1704.04861.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks, 2018. URL https://arxiv.org/abs/1608.06993.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

Johnson, T. B., Agrawal, P., Gu, H., and Guestrin, C. Adascale sgd: A user-friendly algorithm for distributed training, 2020. URL https://arxiv.org/abs/2007.05105.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima, 2017. URL https://arxiv.org/abs/1609.04836.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lau, T. T.-K., Li, W., Xu, C., Liu, H., and Kolar, M. Adaptive batch size schedules for distributed training of language models with data and model parallelism, 2025. URL https://arxiv.org/abs/2412.21124.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets, 2018. URL https://arxiv.org/abs/1712.09913.

Liao, L., Li, H., Shang, W., and Ma, L. An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Trans. Softw. Eng. Methodol.*, 31(3), April 2022. ISSN 1049-331X. doi: 10.1145/3506695. URL https://doi.org/10.1145/3506695.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the variance of the adaptive learning rate and beyond, 2021. URL https://arxiv.org/abs/1908.03265.

McCandlish, S., Kaplan, J., Amodei, D., and Team, O. D. An empirical model of large-batch training, 2018. URL https://arxiv.org/abs/1812.06162.

Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Larochelle, H. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program), 2020. URL https://arxiv.org/abs/2003.12206.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search, 2019. URL https://arxiv.org/abs/1802.01548.

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., and Dahl, G. E. Measuring the effects of data parallelism on neural network training, 2019. URL https://arxiv.org/abs/1811.03600.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size, 2018. URL https://arxiv.org/abs/1711.00489.

Smith, S. L., Elsen, E., and De, S. On the generalization benefit of noise in stochastic gradient descent, 2020. URL https://arxiv.org/abs/2006.15081.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in NLP. In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355. URL https://aclanthology.org/P19-1355/.

Umeda, H. and Iiduka, H. Adaptive batch size and learning rate scheduler for stochastic gradient descent based on minimization of stochastic first-order oracle complexity, 2025. URL https://arxiv.org/abs/2508.05302.

Vogels, T., Karimireddy, S. P., and Jaggi, M. Powersgd: Practical low-rank gradient compression for distributed optimization, 2020. URL https://arxiv.org/abs/1905.13727.

Wu, D., Wang, Y., Xia, S.-T., Bailey, J., and Ma, X. Skip connections matter: On the transferability of adversarial examples generated with resnets, 2020. URL https://arxiv.org/abs/2002.05990.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the transformer architecture, 2020. URL https://arxiv.org/abs/2002.04745.

Xu, H., van Genabith, J., Xiong, D., and Liu, Q. Dynamically adjusting transformer batch size by monitoring gradient direction change, 2020. URL https://arxiv.org/abs/2005.02008.

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes, 2020. URL https://arxiv.org/abs/1904.00962.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning, 2020. URL https://arxiv.org/abs/2001.06782.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization, 2019. URL https://arxiv.org/abs/1901.09321.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning, 2017. URL https://arxiv.org/abs/1611.01578.

A Appendix

Table 4. Training performance with fixed batch size (64) vs. DEBA across six architectures on CIFAR-10 and CIFAR-100. Mean and standard deviation over five random seeds. Green indicates improvement, red indicates degradation.

Architecture	Dataset	Method	Accuracy (%)	Δ Acc	Time (s)	Speedup (%)	Throughput
ResNet-18	CIFAR-10	Fixed	83.05 ± 0.51	–	778 ± 38	–	1.0×
		DEBA	$\mathbf{85.48 \pm 0.66}$	+2.43	512 ± 101	36.6 ± 13.1	1.5×
	CIFAR-100	Fixed	51.81 ± 0.56	–	774 ± 32	–	1.0×
		DEBA	$\mathbf{55.79 \pm 0.77}$	+3.98	441 ± 15	43.0 ± 2.9	1.8×
ResNet-50	CIFAR-10	Fixed	$\mathbf{83.07 \pm 0.62}$	–	1858 ± 28	–	1.0×
		DEBA	82.48 ± 1.62	-0.59	1069 ± 120	43.3 ± 5.5	1.7×
	CIFAR-100	Fixed	50.14 ± 0.81	–	1887 ± 45	–	1.0×
		DEBA	$\mathbf{51.43 \pm 3.85}$	+1.29	1045 ± 81	44.7 ± 3.8	1.8×
DenseNet–121	CIFAR-10	Fixed	84.95 ± 0.57	–	2906 ± 73	–	1.0×
		DEBA	$\mathbf{86.35 \pm 0.68}$	+1.40	1310 ± 197	55.0 ± 6.6	2.2×
	CIFAR-100	Fixed	56.18 ± 0.70	–	2923 ± 46	–	1.0×
		DEBA	$\mathbf{59.42 \pm 0.08}$	+3.24	1097 ± 23	62.4 ± 1.1	2.7×
EfficientNet–B0	CIFAR-10	Fixed	83.18 ± 0.65	–	1443 ± 61	–	1.0×
		DEBA	$\mathbf{85.55 \pm 0.63}$	+2.37	818 ± 210	47.7 ± 13.1	1.8×
	CIFAR-100	Fixed	56.29 ± 0.52	–	1434 ± 56	–	1.0×
		DEBA	$\mathbf{56.72 \pm 0.85}$	+0.43	618 ± 7	56.8 ± 1.5	2.3×
MobileNet-V3	CIFAR-10	Fixed	69.54 ± 3.18	–	882 ± 45	–	1.0×
		DEBA	$\mathbf{76.52 \pm 0.45}$	+6.98	438 ± 9	50.3 ± 3.4	2.0×
	CIFAR-100	Fixed	41.36 ± 0.76	–	899 ± 61	–	1.0×
		DEBA	$\mathbf{45.37 \pm 1.03}$	+4.01	439 ± 3	51.0 ± 3.3	2.0×
ViT-B16	CIFAR-10	Fixed	$\mathbf{68.84 \pm 1.2}$	–	16720.31 ± 139.3	–	1.0×
		DEBA	68.23 ± 0.56	-0.61	15331.05 ± 479.71	8.3 ± 2.89	1.01×
	CIFAR-100	Fixed	38.57 ± 1.1	–	16719.13 ± 293.1	–	1.0×
		DEBA	$\mathbf{39.62 \pm 2.34}$	+1.05	15859.8 ± 331.02	5.1 ± 2.96	1.05×

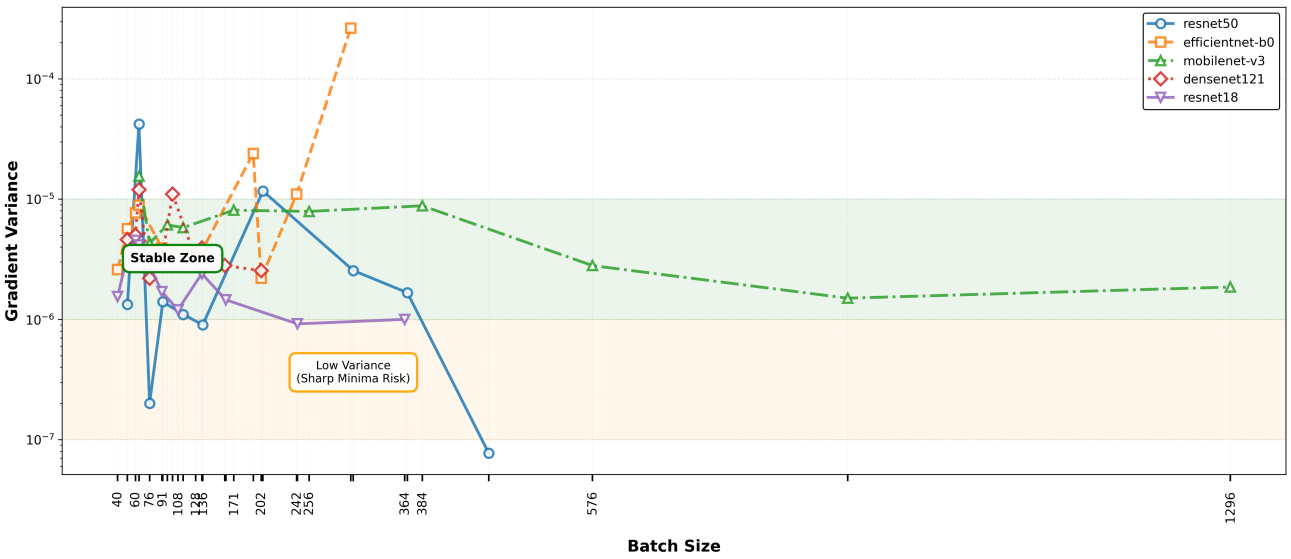

Figure 2. Gradient variance as a function of batch size on CIFAR-10.

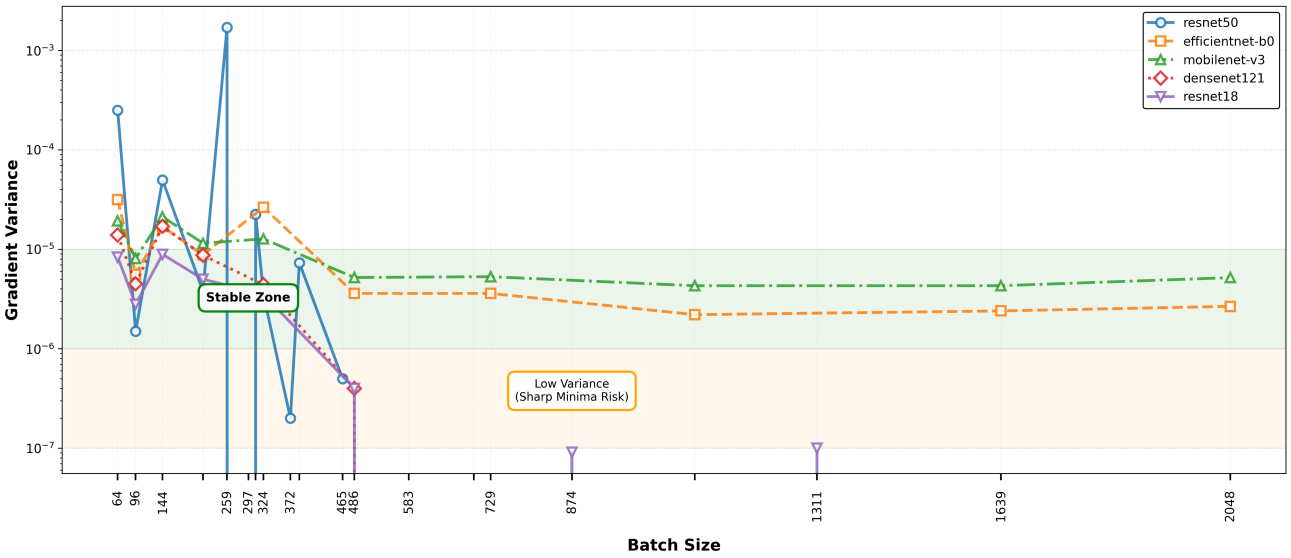

Figure 3. Gradient variance as a function of batch size on CIFAR-100.