Loss Improvement Confidence: A Stability-Based Approach for Adaptive Batch Sizing in Stochastic Gradient Descent

Mohamed Elfadul

University of the Witwatersrand

September 8, 2026

Outlines

1 Motivation

2 The LIC Method

3 Experiments and Results

4 How LIC Adapts the Batch Size

5 Ablation Studies

6 Conclusion

7 References

Motivation

- In large-scale machine learning problems, the problems reduces to solve an optimization problem.
- Given the training data:$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \in \mathbb{R}^d \times \mathcal{Y}$
  - d : the dimension of each input sample.
  - N : the number of the data points.
- Because of N and d are huge, we are interested to think of an optimization method for large-scale ML problems that can handle such big d and N.
- Most of the problems in ML can be written as the following finite sum:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} L_i(\boldsymbol{\theta}). \tag{1}$$

Why SGD

- At the iteration $t+1$ of the problem (1) using Gradient Descent algorithm takes the form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla L(\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t - \frac{\eta_t}{N} \sum_{i=1}^{N} \nabla L_i(\boldsymbol{\theta}_t) \tag{2}$$

- The big drawback: is that computing the gradient at single point involves computing the gradient of the entire sum.
- Solution: to take one sample at time or a random sample of the full set of size B (a mini-batch $\mathcal{B}_t$). Does this idea work?, and why does it work?
- SGD is an unbiased estimate of the true gradient, i.e, in expectation the *stochastic estimate* is the exact gradient. And also, it is the most advanced method that we use currently to train deep learning models proposed by Robbins and Monro in 1951.

The Batch Size Is a Neglected Hyperparameter

- The corresponding SGD update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta_t}{B} \sum_{i \in \mathcal{B}_t} \nabla L_i(\boldsymbol{\theta}_t) \tag{3}$$

- Why is adaptive mini-batch sizing important?
  - Large batch-size implies more stable convergence, and computationally efficient on parallel hardware, often lead to poor generalization.
  - Small batch size implies slow training, and they introduce more noise into the gradient estimate, which can enhance generalization.
  - Adaptive batch sizing aims to balance the efficiency and the performance on unseen data.
- The research question.
  - How can the stability of the loss improvement process be modeled to develop a robust, adaptive mini-batch scheduling algorithm that enhances the model's generalization to unseen data and improves the computational efficiency in deep learning?

Our Position

Idea

Adapt the batch size using the *macroscopic stability* of the training process itself, as measured by the statistical properties of the loss function – not the gradient vector.

Contributions

1. A new stability metric, the **LIC score**, quantifying risk-adjusted progress from the mean and standard deviation of recent loss improvements.
2. A simple, efficient update rule using this score to dynamically adjust the batch size toward a target confidence level.
3. A ten-seed empirical evaluation on MNIST and CIFAR-10 against nine competing schedulers, with full statistical significance testing.
4. We report a systematic one-factor-at- a-time(OFAT) of LIC's Hyper-parameters on both datasets.

A First-Order Model of Epoch-wise Loss Improvement

Let θ_t be the parameters at the start of epoch t, trained with batch size B_t, and define $\Delta\theta_t = \theta_{t+1} - \theta_t$. A first-order Taylor expansion gives the loss change over the epoch:

$$\Delta L_t = L(\theta_t) - L(\theta_{t+1}) \approx -\nabla L(\theta_t)^\top \Delta\theta_t.$$

- A *larger* B_t reduces gradient variance $\Rightarrow$ less volatile $\Delta\theta_t \Rightarrow$ less volatile ΔL_t.
- B_t is therefore a natural **control parameter** for the statistical properties of the loss-improvement process $\{\Delta L_1, \Delta L_2, \dots\}$.

The LIC Score: A Sharpe Ratio for Training

Definition (LIC Score)

Over a sliding window of the past W epochs, with mean loss improvement $\hat{\mu}_{\Delta L,t}$ and its standard deviation $\hat{\sigma}_{\Delta L,t}$:

$$\hat{\mu}_{\Delta L,t} = \frac{1}{W} \sum_{i=t-W+1}^{t} \Delta L_i, \qquad \hat{\sigma}_{\Delta L,t} = \sqrt{\frac{1}{W-1} \sum_{i=t-W+1}^{t} (\Delta L_i - \hat{\mu}_{\Delta L,t})^2}$$

$$C_t = \frac{\hat{\mu}_{\Delta L,t}}{\hat{\sigma}_{\Delta L,t} + \epsilon}$$

- $C_t \gg 1$: progress is large relative to its fluctuations $\Rightarrow$ stable, "confident" trajectory.
- $C_t < 1$: volatility rivals or exceeds average progress $\Rightarrow$ instability.

The Constant Confidence Principle

Principle

Dynamically adjust B_t to keep the measured confidence C_t close to a target C_{target}.

A proportional-integral style controller:

$$\frac{B_{t+1} - B_t}{B_t} = \alpha \left(\frac{C_{\text{target}} - C_t}{C_{\text{target}}} \right) = \alpha \left(1 - \frac{C_t}{C_{\text{target}}} \right)$$

giving the clipped update rule

$$B_{t+1} = \text{clip} \left(\text{int} \left(B_t \cdot \left(1 + \alpha \left(1 - \frac{C_t}{C_{\text{target}}} \right) \right) \right), B_{\text{min}}, B_{\text{max}} \right)$$

- $C_t < C_{\text{target}}$ (below target): term $> 1 \Rightarrow$ batch size grows.
- $C_t > C_{\text{target}}$ (above target): term $< 1 \Rightarrow$ batch size shrinks.
- $\alpha \in (0, 1]$ is the adaptation rate, damping the controller's response.

Algorithm 1: Adaptive SGD with LIC

Input: LR schedule η_t, target confidence C_{target}, window W, rate α

Input: batch bounds $[B_{\min}, B_{\max}]$, epochs T

Initialize θ_0, $B_0 \leftarrow B_{\min}$, loss history queue H_L of size $W + 1$

for $t = 0$ to $T - 1$ **do**

- Train one epoch with batch size B_t; update θ_{t+1}
- Compute validation loss L_{t+1}, append to H_L
- **if** $t + 1 \geq W$ **then**
  - Compute improvements ΔL_i over the window
  - Compute C_t from Eq. (LIC score)
  - Update B_{t+1} via the clipped control rule

Algorithm 1: Adaptive SGD with Loss Improvement Confidence (LIC)

Convergence Guarantee

Standard assumptions: L-smoothness, lower-bounded loss, unbiased stochastic gradient, bounded gradient variance σ^2, and bounded batch size $B_{\min} \leq B_t \leq B_{\max}$.

Theorem (Non-Convex Convergence of LIC-SGD)

Under Assumptions 1–5, running LIC-SGD for T epochs with constant $\eta_t = \eta \leq 1/c$:

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\|\nabla L(\theta_t)\|^2\right] \leq \frac{2(L(\theta_0) - L_{\text{inf}})}{T\eta} + \frac{c\eta\sigma^2}{B_{\min}}$$

- The bound depends on $B_{\min}$, not the (adaptive) trajectory of B_t itself – LIC's clipping guarantees the same asymptotic guarantee as fixed-batch SGD.
- Proof: standard descent lemma + bounded-variance argument, using $B_t \geq B_{\min}$ to control the stochastic gradient noise term.

Datasets and Protocol

- **MNIST**: 54,000 / 6,000 / 10,000 stratified train/val/test split, 30 epochs.
- **CIFAR-10**: 45,000 / 5,000 / 10,000 stratified split, ResNet-18, 200 epochs, with random crop/flip, AutoAugment's CIFAR-10 policy, and Cutout.
- SGD with Nesterov momentum 0.9, weight decay 5×10^{-4}, `MultiStepLR` decayed $10\times$ at 50% & 75% of training.
- **10 random seeds** per configuration (42, 123, 456, 789, 1011, 2048, 3141, 5926, 7890, 9999).
- LIC defaults: $C_{\text{target}} = 1.0$, $W = 5$; $\alpha = 0.1$ on MNIST, $\alpha = 0.05$ on CIFAR-10.

Baselines

Fixed / heuristic

- Fixed BS 128 / 256 / 512
- GNS Scheduler[6]
- AdaBatch Scheduler[2]

Adaptive-sampling (microscopic)

- AdaBatchGrad – Norm_Test[5]
- AdaBatchGrad – IO_Test[5]
- AdAdaGrad – Norm_Test[4]
- AdAdaGrad – IO_Test[4]

Metrics: top-1 accuracy, macro precision/recall/F1, validation & training loss, wall-clock training time.

Statistical Methodology

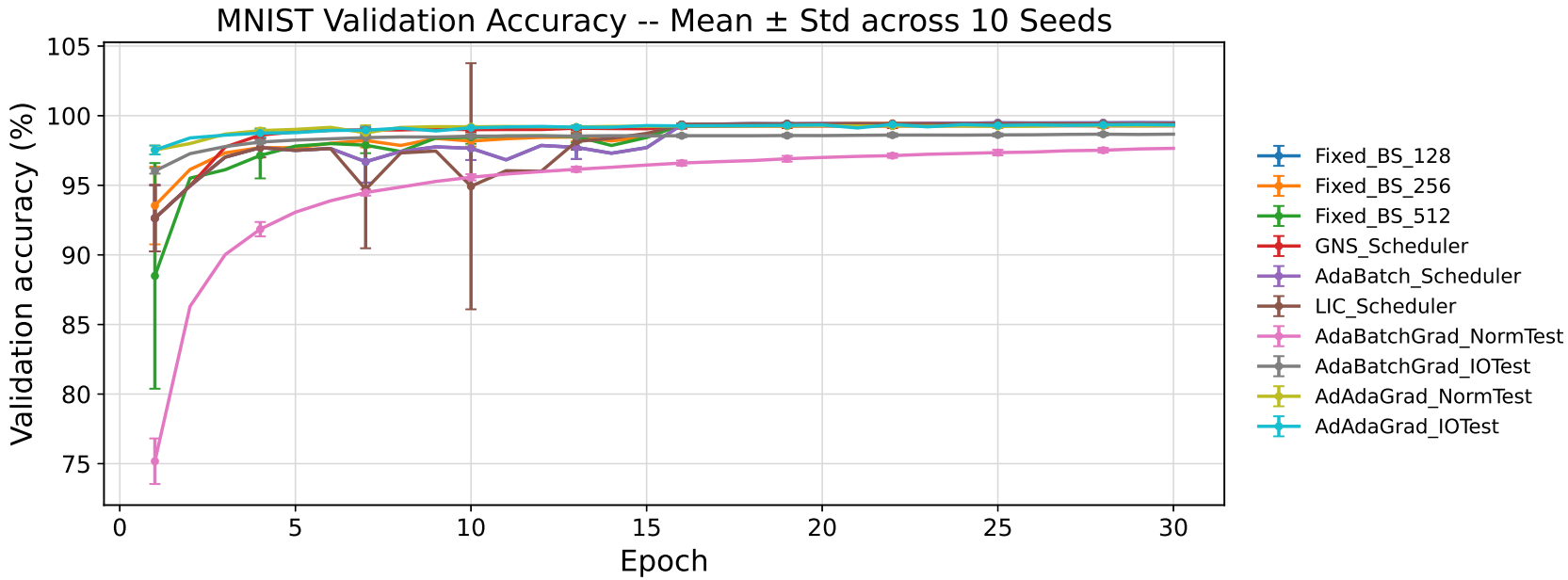
- Per-seed final accuracies compared pairwise against LIC.
- Shapiro-Wilk normality test on the 10 paired differences $\rightarrow$ selects a **paired t-test** (normal) or **Wilcoxon signed-rank test** (non-normal).
- **Bonferroni correction** across the 9 pairwise comparisons.
- Significance coding: $^{***}p_{\text{corr}} < 0.001$, $^{**}p_{\text{corr}} < 0.01$, $^{*}p_{\text{corr}} < 0.05$, ns: not significant.
- **Cohen's d** on paired per-seed differences (signed: positive = LIC scores higher).
- A one-way **ANOVA** across all ten methods on each metric.

MNIST: Classification Performance (Table 1)

Method	Test Acc. (%)	F1 (%)	Sig. (d)
LIC Scheduler	**99.525 ± 0.051**	**99.522 ± 0.052**	–
Fixed_BS_128	99.536 ± 0.045	99.533 ± 0.045	ns (−0.17)
Fixed_BS_256	99.516 ± 0.052	99.513 ± 0.052	ns (0.10)
Fixed_BS_512	99.440 ± 0.056	99.435 ± 0.057	* (1.24)
GNS Scheduler	99.300 ± 0.088	99.295 ± 0.089	*** (2.35)
AdaBatch Scheduler	99.530 ± 0.040	99.527 ± 0.041	ns (−0.07)
AdaBatchGrad_NormTest	97.904 ± 0.065	97.892 ± 0.067	*** (19.24)
AdaBatchGrad_IOTest	98.772 ± 0.095	98.765 ± 0.096	*** (7.04)
AdAdaGrad_NormTest	99.321 ± 0.074	99.315 ± 0.076	*** (2.27)
AdAdaGrad_IOTest	99.422 ± 0.070	99.417 ± 0.070	* (1.22)

All methods except AdaBatchGrad occupy a tight 99.30–99.54% band.

MNIST: Validation Accuracy Trajectories



- Stability: LIC has the *largest* validation-loss std among the nine competitive methods (0.0696 ± 0.0318), driven by 2 of 10 seeds reaching elevated validation loss while final accuracy stays normal for those same seeds.

MNIST: Training Dynamics and Average Time (Table 2)

- AdaBatchGrad_IOTest: **10.11 min** average training time – 2.4–3.2× the cost of the competitive group (3.08–4.17min), driven by per-step adaptive-sampling test overhead, not the batch-size trajectory itself.

Method	Val. Loss	Train Loss	Time (min)
LIC_Scheduler	0.0696 ± 0.0318	0.0672 ± 0.0124	**3.49 ± 0.04**
Fixed_BS_128	0.0585 ± 0.0056	0.0723 ± 0.0124	4.15 ± 0.03
Fixed_BS_256	0.0467 ± 0.0063	0.0685 ± 0.0152	3.34 ± 0.03
Fixed_BS_512	0.0548 ± 0.0130	0.0812 ± 0.0214	3.13 ± 0.03
GNS_Scheduler	0.0407 ± 0.0066	0.0622 ± 0.0143	3.29 ± 0.04
AdaBatch_Scheduler	0.0586 ± 0.0056	0.0723 ± 0.0124	**4.13 ± 0.05**
AdaBatchGrad_NormTest	0.1786 ± 0.0068	0.2655 ± 0.0074	**6.68 ± 0.05**
AdaBatchGrad_IOTest	0.0511 ± 0.0021	0.0733 ± 0.0025	**10.11 ± 0.70**
AdAdaGrad_NormTest	0.0299 ± 0.0021	0.0304 ± 0.0012	3.10 ± 0.03
AdAdaGrad_IOTest	0.0298 ± 0.0037	0.0279 ± 0.0012	**3.74 ± 0.23**

MNIST: Takeaways

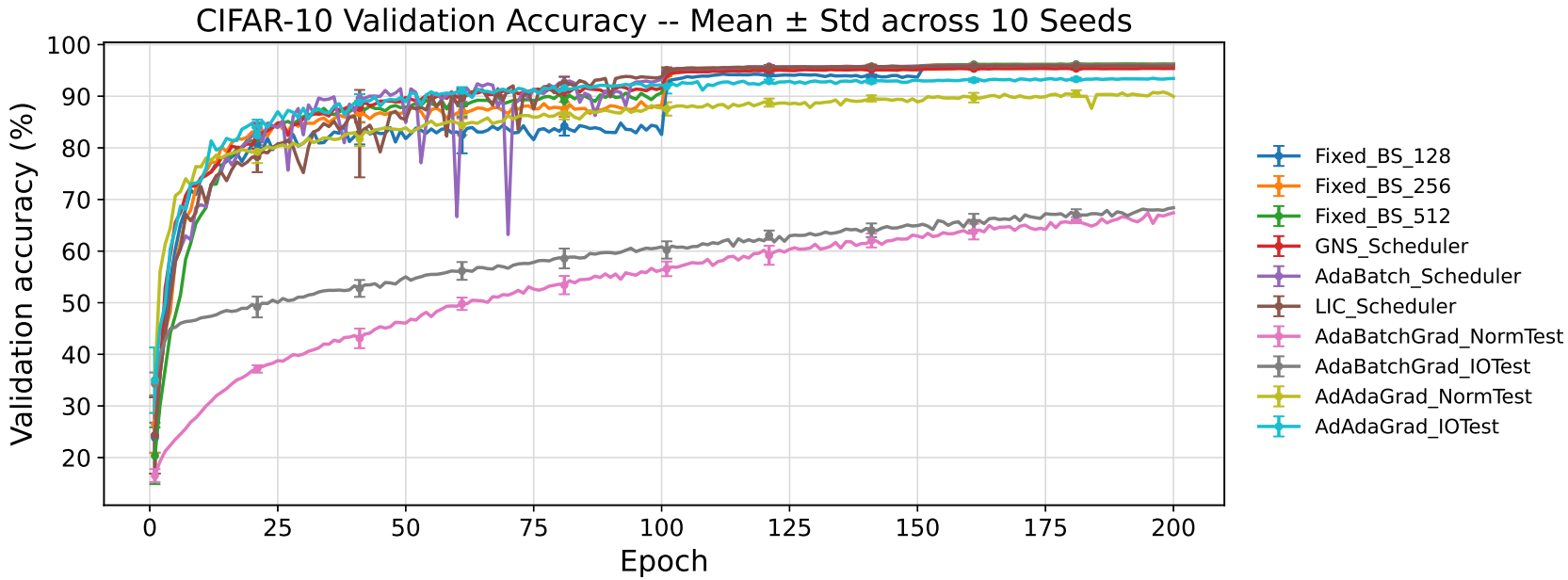
- LIC (99.525%) is **statistically indistinguishable** from Fixed_BS_128, Fixed_BS_256, and AdaBatch Scheduler.
- LIC **significantly exceeds** Fixed BS 512 (+0.085 pts, $d = 1.24$) and AdAdaGrad IOTest (+0.10 pts, $d = 1.22$) – small but real margins given the tight seed-to-seed variance (≤ 0.09 pts for 8/10 methods).
- LIC is **significantly ahead** of all four adaptive-sampling (AdaBatchGrad / AdAdaGrad) methods.
- Claim: LIC is *competitive with*, not superior to, the best-tuned fixed/heuristic schedules on MNIST.

CIFAR-10: Classification Performance (Table 3)

Method	Test Acc. (%)	F1 (%)	Sig. (d)
LIC Scheduler	**95.640 ± 0.229**	**95.636 ± 0.228**	–
Fixed_BS_128	95.781 ± 0.136	95.781 ± 0.135	ns (−0.50)
Fixed_BS_256	95.932 ± 0.156	95.931 ± 0.158	** (−1.79)
Fixed_BS_512	95.774 ± 0.174	95.772 ± 0.175	ns (−0.40)
GNS Scheduler	94.796 ± 0.202	94.795 ± 0.201	*** (3.08)
AdaBatch Scheduler	95.617 ± 0.150	95.616 ± 0.151	ns (0.08)
AdaBatchGrad_NormTest	66.313 ± 1.008	66.008 ± 0.938	*** (30.44)
AdaBatchGrad_IOTest	67.414 ± 1.117	67.181 ± 1.049	* (26.42)†
AdAdaGrad_NormTest	89.492 ± 1.101	89.469 ± 1.172	*** (4.98)
AdAdaGrad_IOTest	92.913 ± 0.255	92.900 ± 0.257	*** (9.85)

†Wilcoxon signed-rank test used (Shapiro-Wilk rejected normality of paired differences).

CIFAR-10: Validation Accuracy Trajectories



- **Stability**: our LIC method has shown more smooth curve on the hard dataset CIFAR-10, the harder the dataset gets, the more smoother our method will perform.

CIFAR-10: Training Dynamics and Average Time (Table 4)

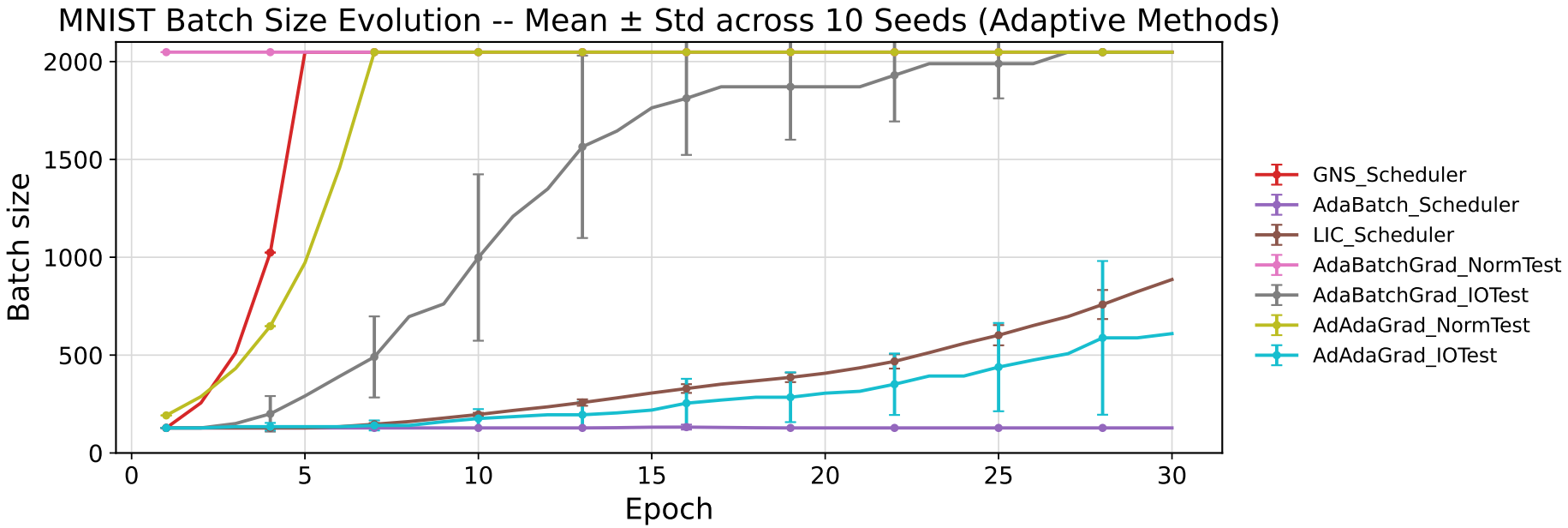
- AdaBatchGrad NormTest / IOTest: 66.3% / 67.4% accuracy – **28–29 points below** LIC and every fixed-batch baseline, while taking $\approx 2.7\times$ longer to train (135.6 / 139.4 min vs. 49–56 min for the competitive group).
- AdAdaGrad variants drop from a 99.3–99.4% MNIST band to 89.5% (NormTest) / 92.9% (IOTest).

Method	Val. Loss	Train Loss	Time (min)
LIC_Scheduler	0.336 ± 0.051	0.351 ± 0.018	$\mathbf{51.1 \pm 0.4}$
Fixed_BS_128	0.372 ± 0.007	0.482 ± 0.006	53.9 ± 0.7
Fixed_BS_256	0.316 ± 0.007	0.404 ± 0.007	68.3 ± 51.7
Fixed_BS_512	0.310 ± 0.014	0.318 ± 0.015	$\mathbf{49.8 \pm 0.4}$
GNS_Scheduler	0.294 ± 0.016	0.335 ± 0.020	52.4 ± 0.5
AdaBatch_Scheduler	0.323 ± 0.023	0.331 ± 0.006	$\mathbf{50.5 \pm 0.4}$
AdaBatchGrad_NormTest	1.280 ± 0.014	1.532 ± 0.014	136.7 ± 0.5
AdaBatchGrad_IOTest	1.130 ± 0.031	1.409 ± 0.029	140.4 ± 0.5
AdAdaGrad_NormTest	0.428 ± 0.014	0.522 ± 0.022	51.7 ± 0.3
AdAdaGrad_IOTest	0.336 ± 0.008	0.336 ± 0.007	56.6 ± 1.4

CIFAR-10: Takeaways

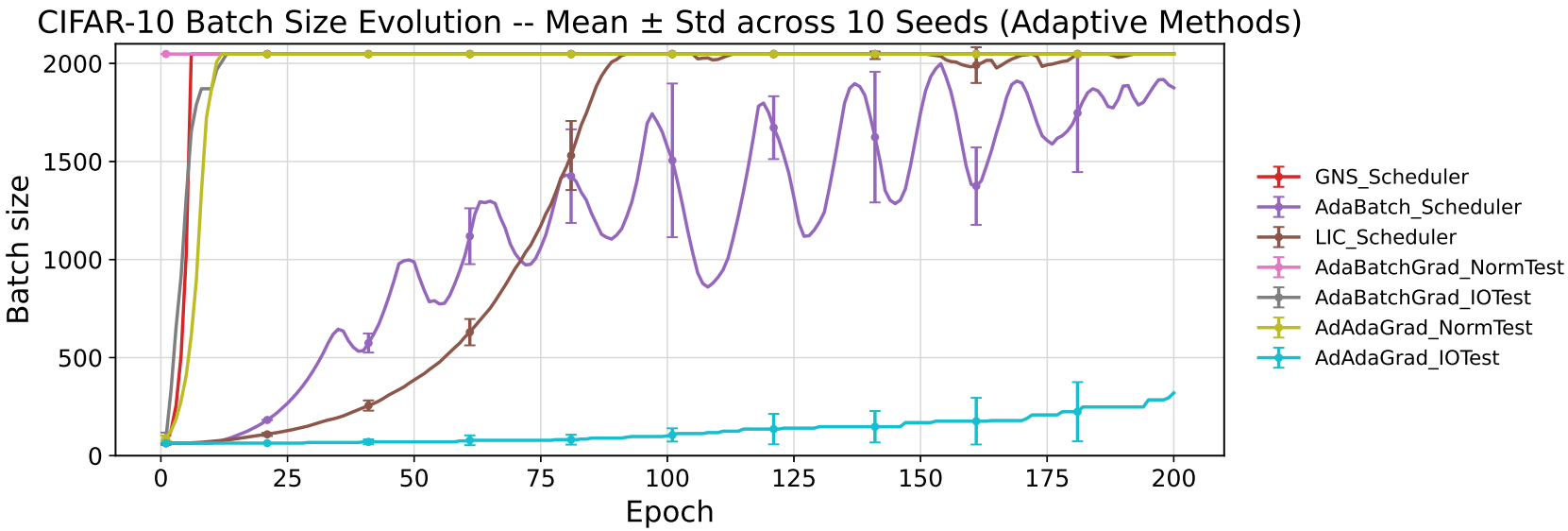
- LIC is **not significantly different** from Fixed_BS_128, Fixed_BS_512, or AdaBatch Scheduler ($p_{\text{corr}} = 1.0$ in every case).
- LIC is **significantly, but only modestly**, behind the single best fixed setting, Fixed BS 256 (95.932% vs. 95.640%, a 0.29-point gap, $p_{\text{corr}} = 0.0028$, $d = -1.79$). *An honest result – we do not explain it away.*
- LIC **beats every one** of the four AdaBatchGrad / AdAdaGrad variants by a large, significant margin (0.84 to 29.3 accuracy points).
- LIC is competitive with the best fixed schedule *in most cases*, without manual batch-size tuning, and generalizes more robustly across dataset difficulty than the tested adaptive-sampling methods.

MNIST: Batch-Size Evolution



- When you deal with a relatively simple dataset like the MNIST, don't expect our LIC method to use large batch sizes, and that's should let you expect less time will be needed for the training.

CIFAR-10: Batch-Size Evolution



- When you deal with a harder dataset like the CIFAR-10, expect our LIC method to gradually use large batch sizes, which in turn, allow the training time to increase.

One-Factor-at-a-Time (OFAT) Ablation Design

- We use $C_{\text{target}} = \{0.5, 0.8, 1, 1.5, 2\}$, and for $W = \{3, 5, 8, 10\}, \alpha = \{0.02, 0.05, 0.1, 0.2, 0.4\}$
- Sweep C_{target}, W, and α individually around a shared default operating point, holding the other two hyperparameters fixed.
- Default: $C_{\text{target}} = 1.0$, $W = 5$; $\alpha = 0.1$ (MNIST) / $\alpha = 0.05$ (CIFAR-10).
- 12 unique configurations per dataset (4 non-default C_{target}, 3 non-default W, 4 non-default α, plus the shared default), each over the same 10 seeds.

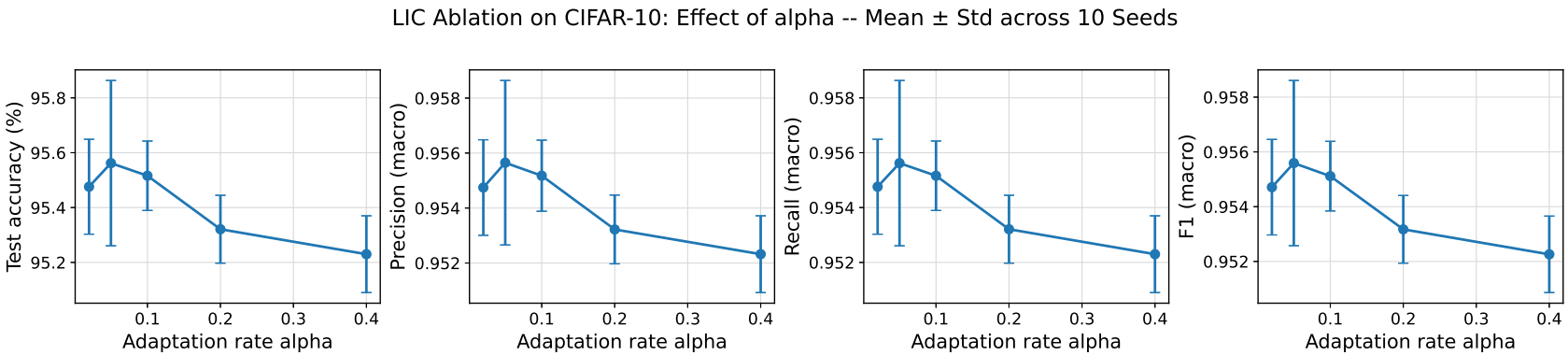
MNIST Ablation: Accuracy Is Flat

- Mean test accuracy varies by **less than 0.05 percentage points** across all three sweeps (C_{target}: 99.497–99.539%; W: 99.524–99.549%; α: 99.497–99.525%).
- Every configuration's mean falls within roughly one pooled standard deviation of every other configuration's mean in the same sweep.
- Training time *does* respond to α: from **3.73 min** ($\alpha = 0.02$) down to **3.33 min** ($\alpha = 0.4$), an $\approx$11% reduction, with *no* accuracy penalty.
- Interpretation: on an easy, near-saturated task, LIC's own hyperparameters barely matter for accuracy – a practical advantage over tuning a fixed batch size.

CIFAR-10 Ablation: Three Distinguishable Patterns

1. **Low** $C_{\text{target}} = 0.5$: inflates seed-to-seed std by 4–8$\times$ (± 1.280 vs. ± 0.15–0.65 elsewhere) – a lax confidence threshold lets small batch sizes persist longer in some seeds, amplifying variance.
2. **Larger** W ($3 \rightarrow 10$): mildly *reduces* std ($\pm 0.251 \rightarrow \pm 0.147$), consistent with $1/\sqrt{W}$ scaling of a sliding-window standard error.
3. **Larger** α (≥ 0.1): mean accuracy declines monotonically, 95.52% ($\alpha = 0.1$) $\rightarrow$ 95.23% ($\alpha = 0.4$), a 0.29-point drop – the mirror image of the MNIST time/accuracy trade-off; wall-clock time stays flat ($\approx$51–52 min).

CIFAR-10 Ablation: Effect of The Adaptation Rate α



Honesty Points

- LIC is **not uniformly superior** to well-tuned fixed schedules on either dataset – competitive, not dominant.
- On CIFAR-10, the best fixed setting (Fixed_BS_256) reliably beats LIC by a small but real 0.29-point margin.
- LIC shows a validation-loss / accuracy **dissociation** on MNIST: its largest validation-loss std among competitive methods does not translate into worse accuracy for the affected seeds.
- One CIFAR-10 comparison (AdaBatchGrad IOTest) required a Wilcoxon test rather than a paired t-test, giving a weaker statistical strength than the raw effect size alone would suggest.

Conclusion

- LIC: a lightweight, macroscopic controller that adjusts batch size to keep a Sharpe-ratio-style loss-stability signal near a target value.
- Proven to converge to a stationary point under standard non-convex assumptions.
- Empirically: performance **statistically indistinguishable** from most of the strongest fixed and heuristic baselines on both MNIST and CIFAR-10.
- **Significantly and substantially outperforms** the AdaBatchGrad and AdAdaGrad families in accuracy, and, for AdaBatchGrad, in wall-clock cost.
- A competitive to fixed batch-size schedules – though not a uniformly superior one – that generalizes more robustly across dataset difficulty than the tested adaptive-sampling methods.

Future Work

- Apply LIC to other domains (e.g., natural language processing).
- Investigate synergies with adaptive learning-rate optimizers.
- A more sophisticated controller for the batch-size update rule beyond the simple multiplicative approach.

References I



Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le.
AutoAugment: Learning augmentation strategies from data.
In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, 2019.



Aditya Devarakonda, Maxim Naumov, and Michael Garland.
AdaBatch: Adaptive batch sizes for training deep neural networks.
2017.



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.



Tim Tsz-Kit Lau, Han Liu, and Mladen Kolar.
AdadaGrad: Adaptive batch size schemes for adaptive gradient methods.
2024.
arXiv:2402.11215.

References II



Petr Ostroukhov, Aigerim Zhumabayeva, Chulu Xiang, Alexander Gasnikov, Martin Takáč, and Dmitry Kamzolov.
AdaBatchGrad: Combining adaptive batch size and adaptive step size.
2024.
arXiv:2402.05264.



Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le.
An empirical model of large-batch training.
2018.
arXiv:1812.06162.



Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang.
Random erasing data augmentation.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020.

Thank you

Questions?