LOSS UNITS CHANGE THE ALGORITHM: CONSISTENT MULTI-OPTIMIZER AMP

Anonymous authors
Paper under double-blind review

ABSTRACT

Replacing a batch-mean loss with a batch sum, with a compensating learning-rate change, preserves the real-arithmetic SGD update. Under automatic mixed precision (AMP), however, the larger loss can cause one optimizer to skip while another steps. We study how this numerical decision changes multi-optimizer training. *Transactional Covariant AMP* (TC-AMP) adjusts each loss scale inversely to its loss units and uses joint replay: if any block overflows, no optimizer steps and the same minibatch is retried. We prove exact execution equivalence for disjoint SGD and classical-momentum blocks under power-of-two unit changes. The proof assumes matched scaled backward expressions and normal-range arithmetic. In partial DINOv2 fine-tuning on CUB-200-2011, native AMP produces eight partial updates per sum-unit run and different mean–sum model endpoints. TC-AMP matches parameters, predictions, update events, and reference-unit scaler histories between loss units in each of three seeds. It reaches 88.14% development validation accuracy, versus 88.22% for FP32, at $1.94\times$ its committed-image throughput in this cached-feature workload. A per-block replay baseline also matches model endpoints, but requires eight additional retries. Checkpoint interventions show why this distinction matters: saved scaler state alone can change retry cost even when model and optimizer states match. Loss units can thus change either the updates performed or the computation required to perform them. Consistency tests for mixed-precision training should track update decisions and consumed minibatches alongside final weights.

1 INTRODUCTION

A batch sum is a batch mean multiplied by the batch size. With the corresponding learning-rate adjustment, both should implement the same SGD update. More generally, for parameters θ_i updated on loss L_i, rescaling the loss and learning rate by reciprocal factors gives

$$\widetilde{L}_i = c_i L_i, \qquad \widetilde{\eta}_i = \eta_i / c_i, \qquad -\widetilde{\eta}_i \nabla \widetilde{L}_i = -\eta_i \nabla L_i. \tag{1}$$

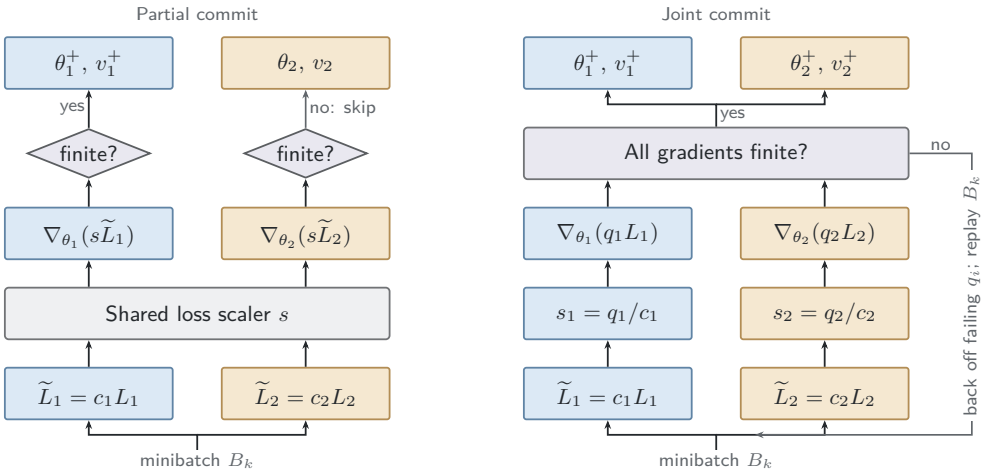
The update formula is unchanged. We ask whether the training program still executes the same updates under automatic mixed precision (AMP).

Consider a feature network and a table of learned class centers, each with its own optimizer, as in Center Loss (Wen et al., 2016). Change only the loss used to update the centers from mean to sum, and divide the center learning rate by the batch size. The larger loss can overflow during backpropagation, before the smaller learning rate is applied. AMP checks the optimizers separately, so it may skip the center update while allowing the network to step (PyTorch Contributors, 2026b). The batch then updates only part of the coupled model. We call this a *partial commit*.

This happens in our CUB-200-2011 fine-tuning runs. Every sum-unit seed has eight such partial commits, and the mean and sum programs finish with different parameters and predictions despite sharing initialization and batch order. One program executes updates that the other skips.

The example exposes two separate issues. First, equivalent loss reductions can trigger different overflow decisions. Second, a local skip can leave a coupled minibatch update half-executed. Each issue needs its own correction. Separate scalers can compensate for independent loss units while still allowing partial commits. Joint replay keeps the blocks together, but may require different numbers of retries across loss units.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Native AMP **(b)** TC-AMP

Figure 1: **Loss scaling and joint updates.** (a) Native AMP may update one block while skipping another. (b) TC-AMP adjusts each scale for its loss units and updates all blocks together, retrying the same minibatch after an overflow. Colors mark blocks; + denotes an update.

Transactional Covariant AMP (TC-AMP) addresses both issues (Figure 1). If a loss is multiplied by c_i, its scale for backpropagation is divided by c_i. If any block overflows, all optimizers wait while the failing scales decrease and the same batch is retried. The first operation aligns the numerical decisions across loss units; the second ensures that every block uses the batch together. For fixed units, inverse-unit initialization of unclamped multiplicative scalers gives the same computation with matching replay and scale-update rules.

Our central distinction is between *changing which updates are applied* and *changing the work required to apply them*. Native partial commits alter the updates. A per-block replay baseline recovers the same CUB model in both loss units, but spends eight extra attempts in each sum-unit run. TC-AMP matches both the model and its update/retry history. The resulting recognition accuracy is 88.14%, against 88.22% for FP32, at $1.94\times$ FP32 throughput for the cached-feature fine-tuning workload. Starting from identical model and optimizer states, we further show that changing only the saved scaler state changes retry cost. Matching the number of successful updates recovers the same model in this intervention.

We give sufficient conditions for equivalent execution with disjoint SGD and classical-momentum blocks, then test the repair and its limits through paired runs, numerical-range controls, and checkpoint interventions.

2 How loss reduction changes optimizer steps

2.1 Loss scaling controls which blocks update

Block $i \in \{1, \ldots, m\}$ is handled by one optimizer, with parameters θ_i, loss L_i, and optimizer state v_i. *Disjoint gradient ownership* means that no parameter belongs to two optimizers and each loss is differentiated only with respect to its own block. The losses can still depend on other blocks' current values, as the network and centers do above.

AMP scales the loss before backpropagation. Gradient unscaling and nonfinite checks precede the optimizer's conditional update (Micikevicius et al., 2018; NVIDIA, 2022; PyTorch Contributors, 2026a). The scaler also retains state: its current scale and a counter used to schedule scale growth. We call one gradient computation on a minibatch an *attempt*. At attempt t, record the nonfinite

checks as

$$M_t = (M_{1,t}, \ldots, M_{m,t}), \qquad M_{i,t} = \mathbf{1}\{g_{i,t} \text{ is nonfinite}\}. \tag{2}$$

Here $g_{i,t}$ is the gradient checked by optimizer i, and $M_{i,t} = 1$ means that the check failed. With optimizer-local step calls, block i updates whenever $M_{i,t} = 0$, regardless of the other blocks (Paszke et al., 2019; PyTorch Contributors, 2026b). The learning-rate correction in Equation 1 acts only if this check allows the step.

An attempt that updates every block is a *joint commit*; one that updates only some blocks is a partial commit. If training then moves to the next minibatch, the skipped blocks never receive their update from the current batch. A changed overflow decision can therefore change which data contribute to each block, even though the SGD formula is unchanged.

2.2 One shared scale cannot offset independent unit changes

Suppose the first loss is unchanged while the second is multiplied by 512. Keeping both scaled backward expressions fixed requires the first scale to stay unchanged and the second to decrease by 512. One common scale cannot do both. More generally:

Proposition 1 (Shared-scalar obstruction)**.** *Suppose one effective scale s is applied to every optimizer-owned loss. Preserving all scaled backward expressions after an independent unit transform requires one $\widetilde{s}$ satisfying $\widetilde{s}c_i = s$ for every i. Such a scalar exists if and only if all c_i are equal.*

Each block requires $\widetilde{s} = s/c_i$, so one value works only when all unit factors agree. The obstruction concerns a single effective scale applied directly to all losses. Allowing a separate unit correction for each block removes it.

2.3 Unit equivalence and joint updates are separate requirements

Two loss representations can make the same partial commits: they agree with each other while leaving the joint update incomplete. Joint replay completes the coupled update, but its overflow decisions can still depend on the loss units. The repair must specify both the scales and what happens after a failed check.

Appendix B demonstrates possible consequences in quadratic examples: repeated partial updates reverse stability, and data-correlated skipping shifts the expected fixed point. The neural-network experiments measure the actual update events, resulting models, and retry costs.

3 Transactional Covariant AMP

TC-AMP first cancels the loss-unit factor in each block's scaled expression, then commits or retries the minibatch as a whole.

3.1 Keep loss scales in a reference unit

Take L_i as the reference loss and $c_i L_i$ as its rescaled representation. Block i keeps a reference scale q_i. For the rescaled loss, it uses the effective scale

$$\widetilde{s}_i = q_i/c_i. \tag{3}$$

The unit factor then cancels before differentiation:

$$\widetilde{s}_i \widetilde{L}_i = (q_i/c_i)(c_i L_i) = q_i L_i. \tag{4}$$

When this cancellation is exact and both programs form the same floating-point expression, their backward tensors and nonfinite checks agree. The scaler updates q_i and derives the effective scale as q_i/c_i. We denote its reference-unit state by Q_i.

For fixed c_i, multiplicative growth and backoff commute with division by c_i. An unclamped per-block scaler initialized at q_i/c_i therefore implements the same computation with matching replay and scale-update rules. Reference units give a common basis for comparing saved scaler states.

3.2 One minibatch, one transaction

Each block computes its gradient separately. The step decision is joint:

1. If every block is finite, unscale the gradients, step all optimizers, and advance the batch index and all state tied to a successful update.
2. If any block is nonfinite, step no optimizer. Reduce the failing blocks' scales and retry the same minibatch, leaving parameters, optimizer buffers, batch index, schedulers, projections, and exponential moving averages unchanged.

Let b_i be block i's growth counter. On failure, nonfinite blocks reduce q_i and reset b_i; finite blocks leave both unchanged. Failed attempts advance the attempt and replay counters. On a joint commit, every b_i advances once and the common growth rule updates q_i. Both loss representations use these rules. Any scale clamp acts on q_i, with effective scale q_i/c_i.

3.3 Equivalence of the complete training state

Weights, optimizer buffers, scaler state, and the next minibatch together determine how training continues. Write the complete state as

$$\mathcal{X}_t = (\theta_t, v_t, Q_t, M_t, k_t, a_t, r_t), \tag{5}$$

where $Q_i = (q_i, b_i, \ldots)$ is the reference-unit scaler state and M_t records the nonfinite checks. The remaining terms track the batch index (or data cursor) k, counters and schedules tied to successful updates a, and replay state r. *Loss-unit covariance* means that the states remain equal in common units after every update or retry. Parameters stay equal; gradient-valued quantities such as momentum rescale with the loss.

Theorem 1 (Transactional loss-unit covariance)**.** *Consider two plain-SGD runs related by Equation 1, with the following conditions:*

1. Shared setup. *Optimizer ownership is disjoint, and each block's gradients are formed separately. The runs share initialization, RNG state, minibatch plan, and deterministic kernels.*
2. Matched expressions. *Each c_i is a power of two. The runs form the same scaled floating-point expression before differentiation, making the unit cancellation exact.*
3. Matched state transitions. *Each reference-unit scaler state is updated using only its own state and nonfinite mask, with no representation-dependent clamp. Both runs use the transaction above.*

The complete state trajectories are then bitwise related by the loss-unit transformation at every attempt and joint commit, while corresponding binary operations remain in a common normal representable regime.

Equation 4 makes the scaled gradients and nonfinite masks identical. Joint replay preserves the state relation on failure; reciprocal learning rates preserve it on a successful update. Appendix A gives the induction and the minibatch-dependent-unit extension.

For classical momentum, the buffer transforms as $\widetilde{v}_i = c_i v_i$. The result extends to zero-initialized buffers with fixed μ, zero dampening, and normal finite intermediates (Appendix A). Scaling and rounding can cease to commute at float32 subnormal values; Section 5.5 examines this limit (Goldberg, 1991; Higham, 2002; IEEE, 2019).

4 Experimental setup

Paired programs. Each method runs a mean–sum pair with the compensating learning-rate change. The unit factor is $c = B$ for Center Loss and cross-entropy. Pairs share initialization, batch order, and deterministic kernels. *Agreement* compares the two loss units within one method. FP32 supplies the non-AMP reference for paired execution, accuracy, and throughput.

Table 1: **What changes between methods.** $s_0 = 65{,}536$ is the default initial scale, and c_i is block i's loss-unit factor. Local skip decides each optimizer separately; joint replay retries the entire minibatch.

Method	Loss scales	Overflow response	Initial effective scale
FP32	None	—	—
NATIVE	Shared	Local skip	s_0
Independent scalers	Per block	Local skip	s_0
ATOMIC-SHARED	Shared	Joint replay	s_0
ATOMIC-BLOCK	Per block	Joint replay	s_0
TC-AMP	Per block	Joint replay	s_0/c_i

We compare paired models, update/retry sequences, and reference-unit scaler states. Model agreement uses prediction agreement, reference-unit momentum, and relative parameter discrepancy

$$d_\theta = \frac{\|\theta_{\text{sum}} - \theta_{\text{mean}}\|_2}{\|\theta_{\text{mean}}\|_2}. \tag{6}$$

Validation accuracy measures task quality. Throughput counts images from successful joint updates and includes the time spent on failed attempts.

CUB fine-tuning. On CUB-200-2011, we train the final two DINOv2-base blocks and task heads using cached tokens from a frozen prefix (Wah et al., 2011; Oquab et al., 2023). The trainable network and class-center table use separate optimizers. A deterministic 4,794/1,200 train/validation split comes from the official training set; the official test split is unused. Each of the six methods below runs both center-loss units for three seeds.

Comparisons. Table 1 separates scale sharing, local versus joint step decisions, and initialization across loss units. In particular, ATOMIC-BLOCK starts both representations at the same effective scale; TC-AMP starts them at the same reference-unit scale. Their replay and scale-update rules are otherwise identical.

All scalers use backoff factor $1/2$, growth factor 2, and growth interval 2,000, with uncapped scales. Framework scalers retain their optimizer-local success counters. Joint-replay methods update their growth counters and schedules as specified in Section 3.

Budgets and diagnostic workloads. CUB uses $B = 512$ and 540 scheduled minibatches. Native and independent scalers attempt each batch once, including batches with partial commits. Replay methods complete 540 joint commits; extra attempts measure retry cost.

The CIFAR-10 Center-Loss and CIFAR-100 cross-entropy experiments use a GroupNorm ResNet-18 (He et al., 2016; Wu & He, 2018; Krizhevsky, 2009). Each uses disjoint optimizer blocks and 2,100 scheduled minibatches in each of three seeds. The ResNet and CUB runs use FP16 autocast unless stated, no stochastic augmentation, deterministic kernels, and disabled TF32. Where momentum is used, it is classical momentum 0.9, with zero dampening, no Nesterov transform, and no weight decay. Appendices C and D give full settings and gradient ownership.

5 RESULTS

The experiments test how loss reduction changes training, how scaling and replay restore agreement, and how saved scaler state affects continued training.

5.1 EQUIVALENT LOSS REDUCTIONS PRODUCE DIFFERENT CUB MODELS

Native AMP ends with different mean–sum models in all three CUB seeds (Table 2). Each sum-unit run has eight partial commits: the network updates while the center optimizer skips. Independent scalers produce the same eight partial commits and model discrepancies because their step decisions remain optimizer-local.

Table 2: **Mean–sum agreement in CUB fine-tuning.** Ranges cover three seeds; the last two columns count pairs with different update events or final parameters.

Method	d_θ	Prediction agreement	Different events (seeds)	Different parameters (seeds)
FP32	0	1.000	0	0
NATIVE	.062–.086	.975–.983	3	3
Independent scalers	.062–.086	.975–.983	3	3
ATOMIC-SHARED	.008–.011	.994–.998	3	3
ATOMIC-BLOCK	0	1.000	3	0
TC-AMP	**0**	**1.000**	**0**	**0**

Table 3: **Loss scaling and joint replay on CIFAR-10.** Events are partial commits for native AMP and retries for joint-replay methods, shown as mean $\rightarrow$ sum across three seeds. Scaler states are compared in reference units.

Method	Events: mean $\rightarrow$ sum	d_θ	Prediction agreement	Model / scaler match
FP32	$[0, 0, 0] \rightarrow [0, 0, 0]$	0	1.000	yes / yes
NATIVE	$[2, 1, 1] \rightarrow [11, 11, 12]$	.034–.041	.647–.675	no / no
ATOMIC-SHARED	$[2, 1, 1] \rightarrow [11, 11, 12]$	.033–.040	.590–.721	no / no
ATOMIC-BLOCK	$[2, 1, 1] \rightarrow [12, 11, 12]$	0	1.000	yes / no
TC-AMP	$[2, 1, 1] \rightarrow [2, 1, 1]$	**0**	**1.000**	**yes / yes**

Joint replay removes partial commits. A shared scaler (ATOMIC-SHARED) still produces different mean–sum models. Per-block scalers (ATOMIC-BLOCK) recover bitwise-equal parameters and reference-unit momentum. The sum runs require eight extra retries: 548 attempts in sum units and 540 in mean units. Their saved scaler states also differ.

TC-AMP matches the paired models and update/retry sequences in all three seeds. Every run completes in 540 attempts, with exact agreement in reference-unit momentum and scaler state. Section 5.4 isolates the effect of saved scaler state on continued training.

Task quality and efficiency. TC-AMP reaches 88.14% mean development validation top-1 accuracy, compared with 88.22% for FP32 and 88.14% for ATOMIC-BLOCK. Accuracy is identical across loss units in every seed.

On one A6000, TC-AMP processes 900.7 jointly committed images/s, $1.94\times$ the FP32 rate and within 0.4% of native AMP. These measurements cover the trainable DINOv2 tail and heads with cached frozen-layer tokens. Successful attempts require one forward/backward pass; replay adds work only after an overflow. Across the CIFAR workloads, paired overhead relative to native AMP is 0.13% for Center Loss, 0.17% with momentum, and 0.24% for cross-entropy. Appendix D reports per-method accuracy, attempts, replays, throughput, and memory.

5.2 WHAT SCALING AND REPLAY EACH REPAIR

In the three-seed CIFAR-10 Center-Loss comparison, joint replay removes partial commits (Table 3). With a shared scaler, mean and sum still produce different models. Per-block replay matches parameters and predictions, while sum units incur more retries and leave different scaler states.

TC-AMP matches parameters, predictions, reference-unit scaler states, and all attempt, commit, and replay counts across loss units.

At $B = 256$, native AMP makes seven partial commits under the standard initial scale, without injected nonfinites. TC-AMP matches models and scaler states (Appendix E).

Cross-entropy stress test. CIFAR-100 with separately formed backbone and head gradients shows the same execution failure under ordinary cross-entropy. Shared and independent scalers produce different models across loss units; TC-AMP matches parameters, predictions, and event

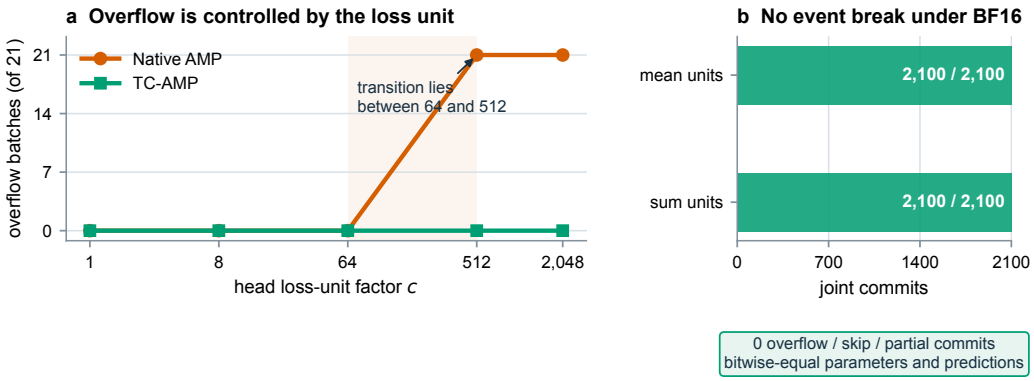


Figure 2: **Numerical-range controls.** (a) Larger loss units trigger native FP16 overflows; TC-AMP remains finite throughout the tested sweep. (b) BF16 without loss scaling completes all 2,100 joint updates with identical mean–sum parameters.

Table 4: **CUB scaler-state intervention, three seeds.** Both runs use sum units. Retries are mean-history / sum-history; discrepancy and prediction agreement are ranges. Matched-update comparisons include parameters and reference-unit momentum.

Attempts	Retries	d_θ	Prediction agreement	State at matched commit count
540	8/0	.00121–.00125	.9967–.9975	bitwise equal
2,048	9/1	.000695–.000727	.9958–.9975	bitwise equal

counts. FP32 accuracy is 1.92%, 0.72%, and 2.08% across seeds, placing this stress test in a low-accuracy regime (Appendix F).

5.3 Changing numerical range switches the failure on and off

To test the overflow mechanism, we vary the loss-unit factor at a fixed model, then change the numerical format (Figure 2). At reference scale 65,536, native FP16 gradients overflow on none of 21 batches for $c \in \{1, 8, 64\}$ and all 21 for $c \in \{512, 2048\}$. The observed transition lies between 64 and 512. TC-AMP has no overflow on these batches at any tested unit factor.

A full-length BF16 control (Kalamkar et al., 2019) uses no loss scaler and produces no overflow-induced skips in this workload. Mean and sum runs each complete 2,100 joint commits with bitwise-equal parameters and identical predictions. The failure appears when the loss-unit change crosses the FP16 overflow threshold and disappears in this BF16 control, linking the observed execution difference to numerical range and conditional steps.

5.4 The same model can require different work after resumption

Does the saved scaler state affect continued training? For each CUB seed, we make two copies of ATOMIC-BLOCK's terminal mean-run checkpoint. Model and optimizer states, schedulers, batch indices, RNG states, and future batches are identical. One copy receives the saved mean-run scaler state; the other receives the saved sum-run state. We call them the mean-history and sum-history runs. Both resume under the same loss representation.

In sum units, the mean-history run starts at the larger scale. After 540 attempts, it has eight retries; the sum-history run has none. After 2,048 attempts, the counts are nine and one. The eight extra retries leave the mean-history run eight successful updates behind, producing different models at the same attempt budget (Table 4). After the same number of successful joint updates, the runs have bitwise-equal parameters and reference-unit momentum. The saved scaler state changes the retry cost of reaching that model.

In mean units, both runs complete 2,048 joint updates with zero retries. Parameters and reference-unit momentum are bitwise identical, and prediction agreement is 1.0. The different scaler states change scale-growth times while leaving every attempt successful.

The paired TC-AMP continuations follow the same update/retry sequence and yield identical predictions at both budgets. Parameters and reference-unit momentum are bitwise equal at 540 attempts. At 2,048 attempts, the remaining differences lie at the float32 subnormal floor, as detailed next.

5.5 Remaining differences come from subnormal rounding

In the CIFAR-10 momentum runs and longer CUB continuation, TC-AMP matches update events and predictions. Residual parameter differences occur at float32 subnormal precision, where scaling and rounding can cease to commute. The CIFAR-10 FP32 pairs also differ at this scale. These differences lie outside the theorem's normal-arithmetic guarantee. Appendices D and G give the measurements and an isolated rounding example.

6 Related work

Loss scaling and equivalent computational graphs. Mixed Precision Training uses FP32 master weights and loss scaling for FP16 training (Micikevicius et al., 2018). Dynamic scaling links nonfinite gradients to scale updates and skipped steps (NVIDIA, 2022; PyTorch Contributors, 2026a); finer-grained scaling addresses underflow within a network (Zhao et al., 2020). Unit Scaling studies scaled computational graphs, including training equivalence under compatible scalings (Blake et al., 2023). We include AMP's conditional branches in this comparison: how loss units affect step decisions, scaler state, and data consumption.

Scale symmetries. Scale-invariant architectures and parameterizations study how parameters, learning rates, and optimizer constants transform under rescaling (Li et al., 2022; Everett et al., 2024); symmetry-compatible optimizer design makes updates equivariant to parameter symmetries (Lau & Su, 2026). Here the loss units change while the intended update stays fixed. This differs from loss weighting in multi-objective training, which changes the optimization problem (Liu et al., 2023).

Execution semantics. PyTorch specifies optimizer-local nonfinite checks and skips (PyTorch Contributors, 2026b); its original AMP design discussion also describes same-batch replay (Carilli, 2019). TC-AMP combines scaling and replay with an explicit loss-unit transformation of scaler and optimizer state. Under Section 3's fixed-unit scale-update conditions, inverse-unit scaler initialization is an equivalent implementation. Our comparisons distinguish model, event, and scaler-state agreement; the checkpoint interventions test the consequence of that distinction.

7 Discussion

Loss scaling can change which parts of a model learn from each minibatch. When optimizers check for overflow independently, one parameter block can update while another skips. This *partial commit* changes the intended coupled update: the batch contributes to only part of the model. Our experiments show that switching between mean and sum losses can trigger this behavior even after adjusting the learning rate to preserve the SGD update. Such events can remain hidden in loss curves and final accuracy. Joint replay preserves each batch's contribution to all blocks. Adjusting each scale for its loss units aligns the overflow decisions across equivalent representations.

Scaler history also affects the computation required to continue training. In our checkpoint interventions, changing only the saved scaler state produced eight additional retries from otherwise identical training states. The runs reached bitwise-identical parameters after completing the same number of joint updates. Scaler state therefore carries information about the cost of reaching a model. Reporting both gradient-computation attempts and completed updates makes this distinction visible when comparing resumed training.

7.1 When the effect matters

The mechanism matters when coupled parameter blocks receive different overflow decisions. When every block remains finite, this source of update mismatch disappears, as observed in our BF16 control. Per-optimizer step and skip logs provide a direct way to check for this condition in FP16 workloads. Their value is greatest when several optimizers are intended to update interdependent parts of a model together.

7.2 Limitations

Our exact equivalence result covers disjoint parameter blocks updated by SGD or classical momentum. It assumes power-of-two rescaling, matched scaled backward expressions, and arithmetic within the normal representable range. Adaptive optimizers, clipping, and weight decay require compatible transformations of their update rules and state.

The experiments isolate the mechanism through controlled classification runs and partial fine-tuning of a pretrained vision model on a single GPU. Its frequency and long-run impact in large-scale training remain open. The reported throughput applies to the cached-feature fine-tuning workload. Each retry adds forward and backward computation, so its cost depends on the workload and overflow frequency.

Our experiments use deterministic replay. With stochastic layers, exact replay requires restoring the random-number state before retrying. Any mutable forward-pass state changed by the failed attempt, including BatchNorm running statistics, must also be restored.

7.3 Future work

An important next question is whether overflow systematically suppresses updates from particular examples or tasks. Our quadratic examples show that data-dependent skipping can shift the optimization fixed point. In neural networks, the key questions are which data are affected and whether repeated omissions change convergence or generalization. Answering them would clarify when numerical failures have lasting consequences for learning.

Preventing these omissions efficiently creates a concrete design problem. Future controllers could choose per-block scales jointly, balancing the preservation of small gradients against the cost of retries. In distributed training, this design would also need to account for the cost of coordinating updates across devices.

These controllers need tests that connect their numerical behavior to the intended algorithm. Equivalent loss reformulations provide a starting point: compare which optimizers step, which batches they use, and how much computation they consume. Extending these tests and their theoretical guarantees to adaptive optimizers and new precision formats would make them useful across a wider range of training systems. The longer-term goal is a practical standard for checking that improvements in training speed preserve the intended updates.

8 Conclusion

Equivalent loss reductions can execute different updates under AMP. TC-AMP matches scales to loss units and replays failed minibatches jointly. CUB fine-tuning demonstrates the resulting consistency at near-native AMP throughput and accuracy close to FP32. The checkpoint interventions show that the same model can require different amounts of computation. Evaluating training equivalence therefore requires tracking the numerical decisions that advance optimization and consume data.

Reproducibility statement

The supplementary material contains the paired programs, configurations, minibatch plans, per-seed results, checkpoints, and scaler traces, along with scripts for the theoretical examples. Appendix C specifies data splits, gradient ownership, and training settings. Recognition metrics use fixed valida-

tion partitions; CUB uses only its official training set, and the small examples use shared training prefixes.

AI USE STATEMENT

We used generative AI tools to clean and reformat the dataset. We did not use them for any other research tasks requiring disclosure (e.g., ideation, experimental design, theory, method implementation, or result interpretation). Additionally, we used AI tools to help debug code and to polish the writing. All AI-assisted outputs were reviewed and verified by the authors, who take full responsibility for this work.

REFERENCES

Charlie Blake, Douglas Orr, and Carlo Luschi. Unit scaling: Out-of-the-box low-precision training. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2548–2576. PMLR, 2023. URL https://proceedings.mlr.press/v202/blake23a.html.

Michael Carilli. Automatic mixed precision for PyTorch. PyTorch design discussion, issue 25081, 2019. URL https://github.com/pytorch/pytorch/issues/25081.

Katie E. Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12666–12700. PMLR, 2024. URL https://proceedings.mlr.press/v235/everett24a.html.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. doi: 10.1145/103162.103163.

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*, pp. 770–778, 2016. URL https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2 edition, 2002. doi: 10.1137/1.9780898718027.

IEEE. IEEE standard for floating-point arithmetic. IEEE 754-2019, 2019.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of BFLOAT16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019. URL https://arxiv.org/abs/1905.12322.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Tim Tsz-Kit Lau and Weijie Su. Symmetry-compatible principle for optimizer design: Embeddings, LM heads, SwiGLU MLPs, and MoE routers. *arXiv preprint arXiv:2605.18106*, 2026. URL https://arxiv.org/abs/2605.18106.

Zhiyuan Li, Srinadh Bhojanapalli, Manzil Zaheer, Sashank Reddi, and Sanjiv Kumar. Robust training of neural networks using scale invariant architectures. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 12656–12684. PMLR, 2022. URL https://proceedings.mlr.press/v162/li22b.html.

Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. FAMO: Fast adaptive multitask optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b2fe1ee8d936ac08dd26f2ff58986c8f-Abstract.html.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

NVIDIA. Train with mixed precision. NVIDIA Deep Learning Performance Documentation, 2022. URL https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision, 2023. URL https://arxiv.org/abs/2304.07193.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

PyTorch Contributors. Automatic mixed precision package: torch.amp. PyTorch Documentation, 2026a. URL https://docs.pytorch.org/docs/stable/amp.html.

PyTorch Contributors. Automatic mixed precision examples. PyTorch Documentation, 2026b. URL https://docs.pytorch.org/docs/main/notes/amp_examples.html.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD birds-200-2011 dataset. Technical report, California Institute of Technology, 2011. URL https://authors.library.caltech.edu/records/cvm3y-5hh21.

Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *Computer Vision – ECCV 2016*, volume 9911 of *Lecture Notes in Computer Science*, pp. 499–515. Springer, 2016. doi: 10.1007/978-3-319-46478-7_31.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision*, pp. 3–19, 2018. URL https://openaccess.thecvf.com/content_ECCV_2018/html/Yuxin_Wu_Group_Normalization_ECCV_2018_paper.html.

Ruizhe Zhao, Brian Vogel, Tanvir Ahmed, and Wayne Luk. Reducing underflow in mixed precision training by gradient scaling. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 2922–2928, 2020. doi: 10.24963/ijcai.2020/404.

A Proof of transactional loss-unit covariance

Assume the complete training states are related at the start of an attempted update. For every block,

$$\widetilde{s}_i \widetilde{L}_i = \frac{q_i}{c_i} c_i L_i = q_i L_i. \tag{7}$$

Under common data, RNG state, deterministic kernels, a common representable regime, and the theorem's common scaled-expression assumption, the scaled forward/backward tensors and nonfinite masks are bitwise identical. Reference-unit scaler states therefore make the same transition.

If any block is nonfinite, joint replay leaves parameters, optimizer state, batch index, and state tied to successful updates unchanged in both runs. The same minibatch is attempted again, preserving the inductive relation. On a finite attempt, let $G_i = q_i \nabla L_i$ be the common scaled gradient. Unscaling the transformed run gives

$$\widetilde{g}_i = \frac{G_i}{q_i / c_i} = c_i \frac{G_i}{q_i} = c_i g_i, \tag{8}$$

and hence

$$\widetilde{\theta}_i^{+} = \theta_i - (\eta_i / c_i)(c_i g_i) = \theta_i - \eta_i g_i = \theta_i^{+}. \tag{9}$$

All blocks commit together and their counters and schedules advance once. Induction over attempts proves Theorem 1.

If $c_{i,t}$ is a specified function of the current minibatch—for example, valid-token count—the proof is unchanged when the effective scale and learning rate are rederived on every attempt and replay recomputes the same $c_{i,t}$ from the same minibatch.

Classical momentum uses the recurrence

$$v_{i,t+1} = \mu v_{i,t} + g_{i,t}, \qquad \theta_{i,t+1} = \theta_{i,t} - \eta_i v_{i,t+1}. \tag{10}$$

Assume $\widetilde{v}_{i,t} = c_i v_{i,t}$. Then

$$\widetilde{v}_{i,t+1} = \mu \widetilde{v}_{i,t} + \widetilde{g}_{i,t} = c_i(\mu v_{i,t} + g_{i,t}) = c_i v_{i,t+1}, \tag{11}$$

$$\widetilde{\theta}_{i,t+1} = \theta_{i,t} - (\eta_i / c_i)(c_i v_{i,t+1}) = \theta_{i,t+1}. \tag{12}$$

The zero buffer supplies the base case. Bitwise induction requires the recurrence and conversion to reference units to remain in the normal regime.

B Partial-commit dynamics

B.1 An isolated miss under contraction

Let the joint map be $z_{t+1} = A z_t + d$ with $\rho(A) < 1$. Replace it once at time τ by $Pz + p$, then return to the joint map. Relative to the joint trajectory,

$$\Delta_{\tau+1+n} = A^n \big[(P - A) z_\tau + (p - d)\big] \longrightarrow 0. \tag{13}$$

Persistence therefore requires repeated partial commits, data-dependent censoring, a basin change, a noncontracting map, or a finite budget.

B.2 Repeated partial commits reverse stability

Consider the quadratic game

$$\min_x \max_y \ \frac{\mu}{2} x^2 + \gamma x y - \frac{\mu}{2} y^2. \tag{14}$$

Write $r = 1 - \eta\mu$ and $h = \eta\gamma$. The joint and y-only maps are

$$A = \begin{pmatrix} r & -h \\ h & r \end{pmatrix}, \qquad C = \begin{pmatrix} 1 & 0 \\ h & r \end{pmatrix}, \tag{15}$$

so alternating joint and y-only updates gives the two-step map

$$CA = \begin{pmatrix} r & -h \\ 2hr & r^2 - h^2 \end{pmatrix}. \tag{16}$$

For $-1 < r < 0$, the joint map is stable when $h^2 < 1 - r^2$, while the periodic map is stable only when

$$h^2 < \frac{(1+r)(1+r^2)}{1-r}. \tag{17}$$

The interval between these bounds is nonempty. Set $\eta = 1$, $\mu = 1.5$, and $\gamma = 0.8$, giving $r = -0.5$ and $h = 0.8$. Then $\rho(A) = 0.943398$ but $\rho(CA) = 1.246888$. Starting from $z_0 = (1, 1)$, 40 cycles give norms 0.01337 and 9624.15, respectively.

B.3 Data-correlated censoring shifts the fixed point

Let

$$\Phi_\xi(x, y) = \frac{a}{2}x^2 + bxy - \frac{a}{2}y^2 + \xi y, \tag{18}$$

where $\xi = 3d$ with probability $1/4$ and $\xi = -d$ with probability $3/4$. Joint updates have expected fixed point $(0, 0)$. If x always commits but y skips exactly when $\xi = 3d$ while the cursor advances, the fixed point is

$$x^* = \frac{bd}{a^2 + b^2}, \qquad y^* = -\frac{ad}{a^2 + b^2}. \tag{19}$$

For $a = b = d = 1$, the shift is exactly $(0.5, -0.5)$; the numerical verification uses $\eta = 0.1$ for both the deterministic expected map and Monte Carlo mean.

C Complete experimental protocol

Table 5: Workloads and comparisons. The first six rows compare mean/sum loss units; the final row varies saved scaler state at a fixed attempt budget.

Workload	Data	Optimizer blocks	c	Steps	Scientific role
Primal–dual constraint	CIFAR-10	classifier / multipliers	16	1,000	constructed witness
Shallow Center Loss	CIFAR-10	network / centers	256	1,500	natural-batch witness
ResNet-18 Center Loss	CIFAR-10	network / centers	2,048	2,100	mechanism decomposition
Center Loss + momentum	CIFAR-10	network / centers	2,048	2,100	optimizer-state boundary
ResNet-18 cross-entropy	CIFAR-100	backbone / head	2,048	2,100	ordinary-objective stress test
CUB DINOv2 Center Loss	CUB-200-2011	trainable tail+heads / centers	512	540	pretrained fine-tuning
CUB scaler-state intervention	CUB development	trainable tail+heads / centers	512	2,048 attempts	effect on retry cost

Table 6: Primary ResNet settings shared by all methods within a workload.

Setting	CIFAR-10 Center Loss	CIFAR-100 cross-entropy
Model	GroupNorm ResNet-18, 128-d embedding, 10 centers	GroupNorm ResNet-18, 128-d embedding, 100-way head
Owned blocks	network / center table	backbone+embedding / classifier head
Split	45,000 train / 5,000 validation	45,000 train / 5,000 validation
Batch / epochs / logical batches	2,048 / 100 / 2,100	2,048 / 100 / 2,100
Network or backbone LR	0.0625	0.0625
Mean auxiliary/head LR	0.125	0.125
Unit transform	center sum, LR$/2048$	head sum, LR$/2048$
Center coefficient	$1/512$	not applicable
Momentum	0 in decomposition; 0.9 in boundary	0.9
Scale-update settings	init 65,536; backoff $1/2$; growth factor 2; interval 2,000	same
Numerics	FP16 autocast; TF32 off; deterministic cuDNN	same
Augmentation	none	none

The Center-Loss mechanism decomposition uses seeds 20260829–20260831. The momentum study uses 2026083001–2026083003. The CIFAR-100 study uses 2026083101–2026083103; the BF16 control uses 2026083191. The paired runs share a fixed 45k/5k train/validation partition and are evaluated after the complete run.

The CIFAR-100 objectives use detached copies to enforce disjoint gradient ownership while preserving the same examples and logits. The backbone loss is mean cross-entropy through detached head parameters. The head loss is cross-entropy through detached embeddings, represented as a batch mean or batch sum. The pair is therefore an exact blockwise learning-rate reparameterization.

D CUB FINE-TUNING AND SCALER-STATE INTERVENTIONS

The CUB experiment uses deterministic 224-pixel center crops of the 5,994 official training images. Within each class, 24 images enter training and six enter validation when available, giving 4,794/1,200 examples. The official 5,794-image test split is counted for dataset integrity but never decoded or evaluated. Frozen DINOv2-base patch embedding and blocks 0–9 produce cached tokens; blocks 10–11, final normalization, a 256-dimensional embedding head, a 200-way classifier, and a 200×256 center table are trainable. The first optimizer owns the trainable DINO tail and heads; the second owns the center table. LayerNorm, the absence of BatchNorm, and exact cached minibatches make a replayed attempt deterministic.

All methods use batch 512 and 60 epochs of nine scheduled minibatches. Native and independent scalers attempt each batch once; transactional configurations retry a failed batch until it jointly commits. Optimization uses classical momentum 0.9, no Nesterov transform or weight decay, and learning rates 10^{-4}, 5×10^{-2}, and 5×10^{-1} for the DINO tail, heads, and centers. The center coefficient is 10^{-2}. The sum representation divides only the center learning rate by 512. Seeds are 2026083101–2026083103. The full matrix contains six methods, two representations, and three seeds (36 runs, 18 paired comparisons). No checkpoint or hyperparameter is selected from the reported validation values.

Table 7: CUB development validation top-1 accuracy (%), mean units / sum units.

Method	Seed 1	Seed 2	Seed 3
FP32	88.50 / 88.50	88.50 / 88.50	87.67 / 87.67
NATIVE	88.50 / 88.50	88.50 / 88.50	87.42 / 88.67
Independent scalers	88.50 / 88.33	88.50 / 88.42	87.42 / 88.50
ATOMIC-SHARED	88.50 / 88.42	88.50 / 88.25	87.42 / 87.42
ATOMIC-BLOCK	88.50 / 88.50	88.50 / 88.50	87.42 / 87.42
TC-AMP	88.50 / 88.50	88.50 / 88.50	87.42 / 87.42

Table 8: CUB training cost, averaged over three seeds. Paired entries are mean / sum. Throughput counts jointly committed images; memory is the maximum peak reserved allocation across paired runs.

Method	Attempts	Partials	Replays	Committed img/s mean / sum	Peak reserved GiB
FP32	540/540	0/0	0/0	465/465	15.96
NATIVE	540/540	0/8	0/0	903/891	12.89
Independent scalers	540/540	0/8	0/0	902/889	12.89
ATOMIC-SHARED	540/548	0/0	0/8	900/889	12.89
ATOMIC-BLOCK	540/548	0/0	0/8	900/888	12.89
TC-AMP	540/540	0/0	0/0	901/900	12.89

Pair-averaged TC-AMP throughput is 0.39% higher than native AMP when counting jointly committed images; its mean-unit throughput is 0.23% lower. Native sum runs complete fewer joint updates in their 540 attempts, whereas replay configurations complete all 540 joint updates.

The mean-unit intervention starts from the terminal ATOMIC-BLOCK mean-run checkpoint for each seed. We copy parameters, centers, optimizer states, scheduler, batch index, RNG, and future batch plan into two runs. One receives the saved mean-run scaler state; the other receives the saved sum-run state. Both complete 2,048 mean-unit joint updates, crossing the 2,000-update scale growth interval with zero retries. All three pairs end with bitwise-equal parameters and reference-unit momentum, and identical validation predictions. The mean-history scale grows from 65,536 to 131,072 after 1,460 commits. The sum-history scale grows from 256 to 512 after 1,523–1,529 commits.

The sum-unit intervention starts from the same terminal mean-run checkpoint. Both ATOMIC-BLOCK runs execute the sum-unit center loss with the corresponding learning-rate and momentum transformations. Each receives exactly 2,048 backward attempts, with a snapshot at 540 attempts. A failed attempt leaves the batch index, scheduler, optimizer, and RNG-dependent batch unchanged. We also save the sum-history run after exactly as many successful updates as the mean-history run has completed. The TC-AMP control starts from the paired terminal TC-AMP checkpoint and runs mean and sum units under the same attempt budgets. All four arms are evaluated for three seeds, giving 12 runs; the official CUB test split remains unused.

Table 4 reports the fixed-attempt results and the states compared at matching commit counts.

The paired TC-AMP runs have identical update/retry sequences, counters, and validation predictions at both milestones. Parameters and reference-unit momentum are bitwise equal at 540 attempts. At 2,048 attempts, their relative discrepancies are 1.50×10^{-44} and 1.23–1.25×10^{-42}, respectively, while prediction agreement remains 1.0. The maximum parameter differences are float32 subnormals. This long continuation reaches the arithmetic boundary characterized in Appendix G, with identical update/retry sequences throughout.

E ADDITIONAL SMALL-SCALE EXPERIMENTS

Table 9: Constructed CIFAR-10 primal–dual example, one paired run. Prediction agreement uses a fixed 2,048-example training prefix.

Method	d_θ	Pred. agreement	Counter equality
FP32	0	1.0000	yes
NATIVE	.006225	.9980	no
ATOMIC-SHARED	0	1.0000	no
ATOMIC-BLOCK	0	1.0000	no
TC-AMP	0	1.0000	yes

The constructed system couples a linear classifier to 16 nonnegative Lagrange multipliers. The sum representation causes four native partial commits; TC-AMP removes the unit-induced event difference and reaches the paired state.

Table 10: CIFAR-10 Center Loss with $B = 256$, one paired run. Prediction agreement uses a fixed 4,096-example training prefix.

Method	Sum partials	Sum replays	d_θ	Pred. agreement
NATIVE	7	0	.003391	.9980
ATOMIC-SHARED	0	7	.002650	.9993
ATOMIC-BLOCK	0	7	0	1.0000
TC-AMP	0	0	0	1.0000

Native AMP updates the network while skipping the center table on seven batches. Shared scaling with joint replay reduces the common scale from 65,536 to 512, producing different rounding across loss units. Per-block replay matches the final parameters with different scaler states and retry counts. TC-AMP also matches these states and counts.

F CIFAR-100 VALIDATION ACCURACIES

Table 11: **CIFAR-100 cross-entropy stress test, three seeds.** Triplets give sum-minus-mean partial commits; discrepancy and prediction agreement are ranges. The last column compares overflow checks, reference-unit scaler states, and update/retry counters.

Method	Sum-minus-mean partial commits	d_θ	Prediction agreement	Event state identical
FP32	$[0, 0, 0]$	0	1.000	yes
One shared scaler	$[11, 8, 10]$	.502–.572	.018–.148	no
One scaler per optimizer	$[10, 13, 7]$	.453–.581	.003–.039	no
TC-AMP	$[0, 0, 0]$	**0**	**1.000**	**yes**

Shared scaling produces $d_\theta = 0.502$–0.572; independent framework scalers produce 0.453–0.581, with prediction agreement as low as 0.26%. TC-AMP completes all 2,100 joint commits with paired replay counts $[3, 3, 4]$, identical reference-unit scaler states and counters, bitwise-equal parameters, and identical predictions. Reference-unit momentum differs by at most four smallest float32 subnormal quanta, the same arithmetic scale present in the FP32 comparator.

Table 12: CIFAR-100 validation accuracy (%), reported as mean units / sum units. The high-batch configuration tests execution equivalence at low task accuracy.

Method	Seed 1	Seed 2	Seed 3
FP32	1.92 / 1.92	0.72 / 0.72	2.08 / 2.08
One shared scaler	2.06 / 3.14	1.26 / 20.22	4.56 / 11.78
One scaler per optimizer	7.84 / 0.68	26.68 / 3.66	1.66 / 2.44
TC-AMP	5.26 / 5.26	3.70 / 3.70	9.56 / 9.56

The fresh-seed BF16 control obtains 10.66 / 10.66% and identical predictions. The comparison measures paired execution and state equality in this low-accuracy regime.

G MOMENTUM SUBNORMAL COUNTEREXAMPLE

In the three-seed momentum study, native sum units add $[11, 11, 12]$ partial commits and yield prediction agreement 0.080–0.406. TC-AMP matches the update/retry sequence, reference-unit scaler states, counters, predictions, and network momentum buffer. Maximum parameter differences are 18, 20, and 3 smallest float32 subnormal quanta, compared with 512, 672, and 2,560 quanta for the FP32 pairs. The following recurrence isolates the rounding effect.

Let $m = 2^{-149}$ be the smallest positive float32 subnormal. Consider classical momentum with $v = 5m$, $g = m$, $\mu = \mathrm{float32}(0.9)$, and unit factor $c = 2048$. In the reference representation,

$$\mathrm{fl}(\mu v + g) = 5m. \tag{20}$$

In the scaled representation, compute $\widetilde{v} = cv$ and $\widetilde{g} = cg$, apply the recurrence, and divide by c. The result is $6m$. Scaling and rounding land on different subnormal lattice points. Replacing m by a normal-scale binary quantum makes the paths bitwise identical. This isolates rounding at subnormal precision from differences caused by skipped updates.

Table 13: Classical momentum, three seeds. TC-AMP matches update decisions and predictions; remaining parameter differences are at float32 subnormal precision.

Method	Sum-minus-mean partial commits	d_θ	Prediction agreement
FP32	$[0, 0, 0]$	8.84×10^{-44}–1.13×10^{-43}	1.000
NATIVE	$[11, 11, 12]$	.380–.443	.080–.406
TC-AMP	$[0, 0, 0]$	6.98×10^{-46}–9.10×10^{-46}	1.000