GIGA: GROUPED SPARSE UPDATES FOR LARGE-BATCH RECOMMENDATION MODEL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling worker count at fixed local batch size improves throughput but coalesces sparse updates, degrading recommendation quality. In click-through rate (CTR) models, this creates a frequency-dependent imbalance: over a given sample budget, frequent IDs lose more update opportunities than rare IDs, so uniform sparse learning-rate scaling cannot correct both head and tail representations. To this end, we propose **Group-wise ID Gradient Aggregation (GIGA)**, which restores within-batch update granularity by treating each active worker group as a virtual sparse update. GIGA sums group means for the parameter update while accumulating their elementwise squares separately in the optimizer state. Group size controls the retained contribution count, affecting highly active IDs while leaving single-worker IDs unchanged. The method requires neither historical frequency statistics, additional collectives, nor persistent per-ID state. With local batch and group size held constant, expected retained contributions are independent of worker count; fusion exactly preserves the group-wise optimizer-state increment while approximating sequential parameter updates. In a controlled 96-worker comparison, GIGA improves AUC and request-level GAUC by **0.03** and **0.13** percentage points, respectively. Combined with dense-parameter scaling, GIGA supports a 16-fold larger global batch of 307,200 examples on 512 workers and delivers $9.8\times$ throughput while matching or improving both metrics relative to the 32-worker baseline. A separate search CTR task confirms gains under both warm-start and from-scratch training. Code is available at `https://anonymous.4open.science/r/giga-review-apache2-339B`.

1 INTRODUCTION

Modern recommendation models use large embedding tables to represent high-cardinality categorical features such as users and items (Naumov et al., 2019; Mudigere et al., 2022). At scale, a common training architecture shards the embedding tables across devices while replicating the dense network, combining model parallelism for sparse parameters with synchronous data parallelism for dense parameters (Mudigere et al., 2022; Zong et al., 2025; Feng et al., 2026). To increase throughput, such systems scale the worker count while retaining a hardware-efficient local batch size; as a result, the global batch grows and a given training sample budget yields fewer optimizer steps. Dense networks can often accommodate this change through learning-rate scaling or longer training (Goyal et al., 2018; Hoffer et al., 2017). Sparse embeddings behave differently because each vector is updated only in batches containing its ID, and ID frequencies can vary by orders of magnitude. Consequently, scaling the global batch creates **frequency-sensitive update coalescing**: more update opportunities are merged into each optimizer step for head IDs than for tail IDs, leaving head-ID embeddings relatively under-updated over the same sample budget and ultimately degrading ranking quality.

This head–tail imbalance stems from the highly skewed access distribution of recommender embeddings, which has also motivated frequency-aware representation and optimization methods (Ginart et al., 2021; Li et al., 2022). As the batch grows, tail IDs are rarely active on multiple workers, so their update counts change little; for head IDs, however, contributions that would form separate smaller-batch updates are increasingly coalesced into one synchronized step. Active-support averaging avoids dilution by inactive workers (Défossez & Bach, 2017), but removes the active-count factor, concentrating the cumulative-update shortfall in the head. A single sparse learning-rate multiplier therefore cannot correct the head without also amplifying the tail, as summarized in Figure 1.

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.

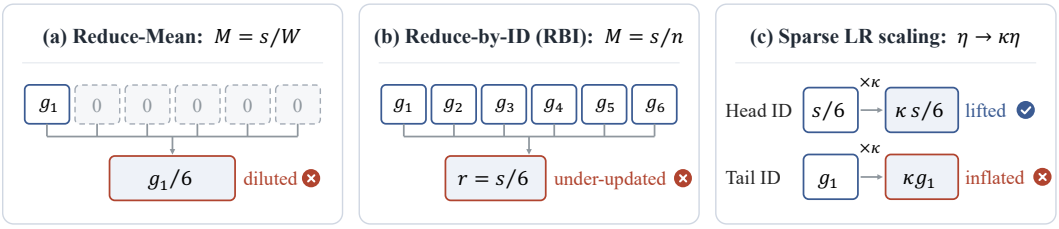


Figure 1: Failure modes of standard sparse-optimization strategies under a large global batch. Here, W is the worker count, n the active-worker count for an ID, and $s = \sum_{w \in S} g_w$. **(a)** Reduce-Mean dilutes a tail-ID update to g_1/W. **(b)** Reduce-by-ID (RBI) avoids this dilution but coalesces n head-ID contributions into one mean update s/n. **(c)** Uniform sparse learning-rate scaling, $\eta \to \kappa\eta$, increases the head-ID update but also inflates the tail-ID update.

This issue is especially relevant to Adagrad, whose coordinate-wise adaptation suits sparse features (Duchi et al., 2011) and which is widely used in industrial CTR and DLRM training (Anil et al., 2022; Liu et al., 2023; Zhang et al., 2025). Under Adagrad, coalescing gradients for the same ID changes both the update numerator and the accumulated squared-gradient state. To our knowledge, existing work on coordinate- or frequency-aware embedding optimization (McMahan et al., 2013; Li et al., 2022; Zheng et al., 2023; Zhu et al., 2024; Li & Lyu, 2026) and distributed synchronization or aggregation (Su et al., 2022; Maleki et al., 2021; Zhang et al., 2025) does not jointly retain ID-dependent contribution granularity in both the sparse update and the Adagrad state. This motivates an aggregation rule that retains both signals.

To address these issues, we propose **Group-wise ID Gradient Aggregation (GIGA)**, a sparse aggregation rule that restores controllable update granularity under large-batch synchronization. GIGA averages each active ID's gradients within worker groups, sums the group means for the parameter update, and accumulates their elementwise squares separately in the Adagrad state. Group size interpolates between active-worker averaging and retaining every active-worker contribution, while leaving single-worker IDs unchanged. For fixed local batch and group size, the expected number of retained contributions over a sample budget is independent of worker count. GIGA reuses the existing sparse all-to-all exchange, requires neither historical frequency statistics nor persistent per-ID state, and introduces no additional collectives. Its fused update preserves the group-wise state increment while approximating sequential Adagrad updates.

We validate GIGA through end-to-end scaling on two industrial CTR workloads. At the same local batch size, scaling active-support averaging from 32 to 96 workers lowers AUC by 0.02 pt and GAUC by 0.10 relative to the 32-worker baseline. At 96 workers, replacing active-support averaging with GIGA raises AUC by **0.03 pt** and GAUC by **0.13 pt**, bringing both above baseline with *all other settings unchanged*. In the full scaling recipe, combining GIGA with dense-parameter scaling supports a 16-fold larger global batch, from 19,200 examples on 32 workers to 307,200 on 512 workers, and delivers $9.8\times$ throughput while matching or improving both baseline metrics. To further assess its generality across tasks and training regimes, we evaluate GIGA on an industrial search CTR model, where it improves both AUC and GAUC under warm-start and from-scratch training.

Contributions. Our contributions are threefold. **(1)** We identify frequency-sensitive update coalescing as a source of large-batch quality loss in sparse recommendation models and explain why uniform learning-rate scaling cannot balance head and tail IDs. **(2)** We introduce GIGA, which controls this coalescing through group-wise gradient means while preserving their separate contributions to the Adagrad state. We analyze how its retained contributions scale with worker count and how its fused update approximates sequential Adagrad; the implementation requires neither historical frequency statistics, additional collectives, nor persistent per-ID state. **(3)** We validate the mechanism on two industrial CTR workloads and scale the global batch 16-fold, achieving $9.8\times$ throughput at baseline-or-better quality, with gains under both warm-start and from-scratch training.

2 Preliminaries

We consider synchronous training with W workers, where worker $w \in \{1, \ldots, W\}$ processes a local minibatch $\mathcal{B}_{w,t}$ of B examples at step t. Embedding tables are sharded across parameter owners,

while dense parameters are replicated across workers. Let $\mathcal{I}_{w,t}$ be the set of embedding IDs occurring in $\mathcal{B}_{w,t}$. For an ID $j \in \mathcal{I}_{w,t}$, let $g_{w,j,t} \in \mathbb{R}^d$ denote its locally aggregated gradient, including mean-loss normalization and any task-specific weighting. We call worker w *active* for j whenever $j \in \mathcal{I}_{w,t}$, irrespective of whether $g_{w,j,t}$ is zero. The workers active for ID j at step t form the set:

$$S_{j,t} := \{w \in \{1, \dots, W\} : j \in \mathcal{I}_{w,t}\}, \qquad n_{j,t} := |S_{j,t}|,$$

where $n_{j,t}$ is the number of active workers. Sparse aggregation is separable across embedding IDs. We therefore fix an arbitrary ID–step pair (j, t) with $n_{j,t} > 0$ and abbreviate $g_w := g_{w,j,t}$, $S := S_{j,t}$, and $n := n_{j,t}$ throughout the single-step analysis. If $n_{j,t} = 0$, no update is applied to ID j at step t. Unless stated otherwise, all vector operations are elementwise. We first specify the baseline sparse Adagrad update and then quantify how global-batch scaling coalesces updates across ID frequencies.

2.1 Sparse Adagrad

Large-scale recommender systems commonly shard embedding tables across parameter owners. During backpropagation, workers send gradients only for rows accessed by their local minibatches, and each owner coalesces the records associated with the same row (Mudigere et al., 2022; Zong et al., 2025). Unlike a dense parameter, which receives a gradient from every worker, an embedding row may be active on only a subset of workers. Averaging its gradient over all W workers would therefore treat absence as a zero contribution and attenuate the result according to the row's activation sparsity. The corresponding owner-side rule, ***Reduce-by-ID*** (RBI), instead averages over the active worker support, following the sparse-normalization principle studied by AdaBatch (Défossez & Bach, 2017). For an embedding vector active on a worker set S of size $n > 0$, let

$$s := \sum_{w \in S} g_w, \qquad r := \frac{s}{n}. \tag{1}$$

Here g_w already aggregates repeated local occurrences. A full-worker mean would be $s/W = (n/W)r$, whereas RBI uses r and thus avoids dilution by inactive workers. In particular, when a row is active on only one worker ($n = 1$), RBI gives $r = g_w$, so the locally aggregated gradient reaches the optimizer without attenuation; averaging over all workers would instead scale it to g_w/W.

The active-support mean r is then passed to the sparse optimizer. We focus on Adagrad, whose coordinate-wise adaptation is well suited to sparse, unevenly activated parameters and is widely used for training industrial recommender models (Duchi et al., 2011; Anil et al., 2022; Liu et al., 2023). Let $e \in \mathbb{R}^d$ and $H \in \mathbb{R}^d_{\geq 0}$ denote the embedding vector and accumulator before the update, and let e^+ and H^+ denote their post-update values. For a vector x, $x^{\odot 2} = x \odot x$ denotes its elementwise square; the square root and division below are also elementwise. Given a learning rate $\eta > 0$ and numerical stabilizer $\epsilon > 0$, RBI followed by sparse Adagrad applies

$$H^+ = H + r^{\odot 2}, \qquad e^+ = e - \eta \frac{r}{\sqrt{H^+} + \epsilon}. \tag{2}$$

RBI avoids inactive-worker dilution but fixes the granularity of optimizer transitions: whenever $n > 0$, it compresses all active-worker gradients into one numerator r, one accumulator increment $r^{\odot 2}$ and one parameter update, irrespective of the support size. Contributions that would form separate updates at a smaller global batch are therefore merged before either the parameter or optimizer state is updated. This coalescing matters little for embeddings that seldom span workers but increasingly affects frequent IDs as the global batch grows. Section 2.2 makes this scaling effect explicit by relating the expected update count to ID frequency and global batch size.

2.2 Update Coalescing

Under RBI, an embedding row receives at most one optimizer transition per global step, regardless of how many workers activate it. Its transition count over a sample budget is therefore governed by the probability of appearing anywhere in the global batch, rather than by its total number of occurrences. Holding the local batch size B constant, increasing the worker count W both enlarges the global batch WB and reduces the number of global steps.

To quantify this effect, suppose each example contains ID j independently with probability $p_j > 0$, and let the sample budget N be divisible by WB. The row is active in a global step with probability

$1-(1-p_j)^{WB}$; considering that processing N examples takes $N/(WB)$ steps, its expected number of optimizer transitions is computed as:

$$U_j(W,B) = \frac{N}{WB}\left[1-(1-p_j)^{WB}\right]. \tag{3}$$

For a rare ID with $p_jWB \ll 1$, the activation probability is approximately p_jWB, so $U_j(W,B) \approx Np_j$ and most occurrences remain in separate global steps. For a frequent ID with $p_jWB \gg 1$, the activation probability approaches one, so $U_j(W,B) \approx N/(WB)$ and many occurrences are coalesced into each step. Increasing W therefore preserves most tail-ID transitions but removes many head-ID transitions. This frequency-dependent loss cannot in general be corrected by a single sparse learning-rate multiplier. Under Adagrad, coalescing also changes the sequence of squared-gradient increments, so numerator rescaling alone cannot recover the optimizer-state trajectory. Appendix A.1 derives the expected counts, while Appendix A.5 discusses local-batch and dense-parameter scaling.

3 Group-wise ID Gradient Aggregation

To address the frequency-dependent update coalescing discussed in Section 2.2, we propose **Group-wise ID Gradient Aggregation (GIGA)**, which introduces tunable sparse-update granularity by forming group-level updates and combining them into a single synchronous optimizer step while preserving their separate state increments. We first define the group-wise virtual updates, then derive their fusion into one optimizer step, analyze the resulting group-size trade-off, and finally describe the distributed implementation for large-scale embedding architectures.

3.1 Group-wise Virtual Updates

We restrict G to a divisor of W and partition the workers into $K = W/G$ disjoint groups $\mathcal{G}_1, \ldots, \mathcal{G}_K$ of equal size G. The same partition is used across embedding rows and training steps. For a given row with active worker set S, only groups containing at least one worker in S contribute to its update. Let n_k denote the number of active workers in group k, let $\mathcal{A}$ collect the nonempty groups, and let A be their count. Each $k \in \mathcal{A}$ then forms an active-support mean $\bar{g}_k$:

$$\begin{aligned} n_k &:= |S \cap \mathcal{G}_k|, \qquad \mathcal{A} := \{k \in \{1,\ldots,K\} : n_k > 0\}, \qquad A := |\mathcal{A}|, \\ \bar{g}_k &:= \frac{1}{n_k}\sum_{w \in S \cap \mathcal{G}_k} g_w, \qquad k \in \mathcal{A}. \end{aligned} \tag{4}$$

Thus, $\bar{g}_k$ is precisely RBI restricted to group k. We regard each nonempty group mean as a ***virtual sparse update*** at batch scale GB. They determine the update granularity but remain virtual: GIGA combines them within a single synchronous optimizer step over the full global batch.

This construction retains one contribution from every active group. Under the activation model in Section 2.2, the expected number of such group–row contributions over N examples is

$$U_j^{\mathrm{virt}}(\boldsymbol{G}, B) := \frac{N}{\boldsymbol{G}B}\left[1-(1-p_j)^{\boldsymbol{G}B}\right]. \tag{5}$$

Unlike the physical-transition count $U_j(W,B)$ in Equation (3), $U_j^{\mathrm{virt}}(G,B)$ has no explicit dependence on W. Holding B and G constant preserves the expected number of retained contributions as workers are added, even though more of them may be fused into each physical step. This count-level invariance motivates the construction but does not imply an invariant parameter trajectory.

3.2 Fused Group-wise Update

The virtual updates capture the desired aggregation granularity, but executing them separately would introduce multiple optimizer transitions within one global step. To this end, GIGA instead fuses them into a single physical transition while retaining their aggregate direction and separate contributions to the optimizer state. Accordingly, define the update numerator M and state increment Q as follows:

$$M := \sum_{k \in \mathcal{A}} \bar{g}_k, \qquad Q := \sum_{k \in \mathcal{A}} \bar{g}_k^{\odot 2}. \tag{6}$$

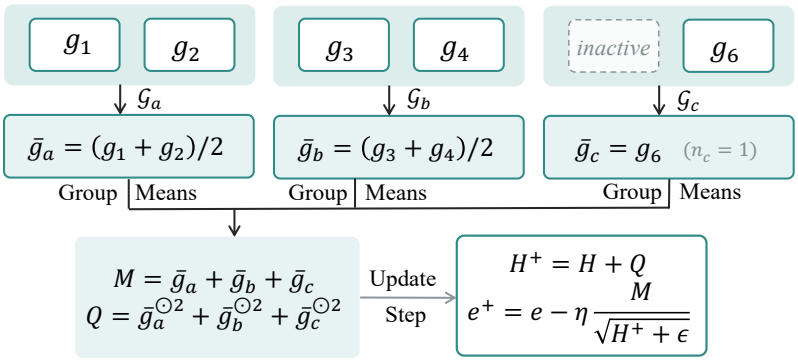


Figure 2: Group-wise sparse aggregation and fused update in GIGA. Workers are partitioned into groups of size $G = 2$, and each nonempty group forms an active-support mean. These means determine M and Q for one Adagrad step; the dashed box marks an inactive worker.

The resulting sparse Adagrad transition, summarized in Figure 2, is

$$\boxed{H^+ = H + Q, \qquad e^+ = e - \eta \frac{M}{\sqrt{H^+} + \epsilon}} \tag{7}$$

For prescribed group gradients, this construction exactly preserves the state increment obtained from sequential accumulation, although it generally produces a different parameter displacement because the final accumulator is applied only once. Section 3.4 gives a coordinatewise comparison and a one-step error bound. The need to construct Q separately is also evident from

$$M^{\odot 2} - Q = 2 \sum_{k<\ell} \bar{g}_k \odot \bar{g}_\ell, \tag{8}$$

where the sum ranges over active-group pairs $k, \ell \in \mathcal{A}$. These cross-group products are absent from separate accumulation and may increase or decrease individual accumulator coordinates. Hence, using $M^{\odot 2}$ instead of Q defines a different state update rather than an equivalent implementation of GIGA. The ablations in Section 4.2 show that this design choice materially affects ranking quality.

3.3 Group-Size Trade-off

Group size controls the balance between preserving fine-grained contributions and limiting the number fused within one physical step. Unlike frequency-aware optimizers that modify learning, clipping, or regularization rules across ID frequencies (Li et al., 2022; Zheng et al., 2023; Zhu et al., 2024), GIGA depends only on the current active support:

$$\begin{aligned} n = 1: &\quad (M, Q) = (g_w, g_w^{\odot 2}) \quad \text{for any } G, \\ G = W: &\quad (M, Q) = (r, r^{\odot 2}) \quad \text{(RBI)}, \\ G = 1: &\quad (M, Q) = \left(s, \sum_{w \in S} g_w^{\odot 2} \right). \end{aligned} \tag{9}$$

Thus, grouping leaves single-worker rows unchanged, recovers RBI when all workers share one group, and retains every active-worker contribution when groups are singletons. For intermediate G, unequal group occupancies reweight worker gradients, so the resulting numerator need not be a scalar multiple of RBI. Nevertheless, under a uniformly random balanced partition, its conditional mean is an activity-dependent rescaling:

$$\mathbb{E}[M \mid S, \{g_w\}_{w \in S}] = \mathbb{E}[A \mid n] r, \tag{10}$$

with assumptions and derivations given in Appendices A.2 and A.3.

Beyond these endpoints, group size balances two forms of aggregation. Smaller G reduces coalescing within each group but may increase the number of group contributions fused into a physical step,

whereas larger G approaches RBI. Because $A \le \min(n, K)$ and $K = W/G$, the group count caps the number of contributions entering a step. For aligned scalar group gradients $\bar{g}_k = g$ and $\epsilon = 0$,

$$|\Delta_{\text{GIGA}}| = \eta \frac{A|g|}{\sqrt{H + Ag^2}}.$$

This special case illustrates how the numerator and accumulator scale jointly; it does not identify an optimal G, which remains an empirical choice (see Appendix C.2).

Distributed Implementation GIGA maps directly onto the existing owner-side sparse routes used by distributed embedding systems (Mudigere et al., 2022; Zong et al., 2025). The receive metadata identifies each record's source worker and hence its preassigned group. Each parameter owner first reduces worker–row records by group and row to obtain $\bar{g}_{j,k}$, then reduces across groups to form (M_j, Q_j). Zero-valued gradients remain in the activity counts whenever the row is present. The implementation requires no additional collective, group-label payload, frequency table, or additional persistent per-ID state. Temporary storage is proportional to the observed group–row pairs and can be bounded by processing complete groups in chunks. Algorithm 1 and Appendix B.1 provide implementation details, while Section 4.3 reports the measured overhead.

3.4 Discussion

Relation to sequential Adagrad. To interpret the fused transition, consider prescribed coordinate-wise group gradients $z_1, \ldots, z_A$ in an arbitrary order, with $H_0 = H$ and $H_a = H_{a-1} + z_a^2$. Sequential Adagrad applies the virtual updates one at a time, so z_a is normalized by the evolving state H_a. GIGA preserves the same total state increment but sums the virtual gradients and applies the final state once:

$$\Delta_{\text{seq}} = -\eta \sum_{a=1}^{A} \frac{z_a}{\sqrt{\boldsymbol{H_a}} + \epsilon}, \quad \Delta_{\text{GIGA}} = -\eta \frac{\sum_{a=1}^{A} z_a}{\sqrt{\boldsymbol{H_A}} + \epsilon}. \tag{11}$$

Thus, both constructions reach the identical accumulator $H_A = H + \sum_a z_a^2$, but generally follow different parameter paths: early sequential updates use smaller denominators, whereas GIGA evaluates every contribution against the final denominator. Appendix A.4 derives the one-step error bound in Equation (A.1). The bound contracts when the prior accumulator dominates the new increment. This comparison is limited to prescribed gradients and absolute error. Cancellation can amplify relative error, and true sequential optimization would recompute gradients after every update.

4 Evaluation

4.1 Experimental Setup

Training setup. Main experiments use a DLRM-like model on a private industrial CTR workload. Runs share a warm-start checkpoint and matched daily datasets, use Adagrad for dense and sparse parameters, and run on PyTorch/RecIS with one AMD MI308X GPU per worker. We write W/B for W workers with local batch size B; $128/600$ therefore uses a global batch of 76,800. Group size G counts workers per GIGA group. Unless stated otherwise, AUC and request-level GAUC differences use the $32/600$ baseline, and quality recovery requires both metrics to match or exceed it.

Baselines We compare GIGA with five sparse aggregation rules: ① Reduce-Mean averages over all workers; ② RBI averages over the active workers for each ID; ③ LR-scaled RBI changes only the sparse learning rate; ④ Reduce-Sum sums active-worker gradients; and ⑤ Capped RBI multiplies RBI's update numerator by $\min(n, c)$, where n is the ID's active-worker count and c caps this multiplier, while retaining RBI's Adagrad state update. Thus, c limits activity-dependent amplification rather than clipping gradient norms. Appendices B.2, B.3 and B.4 provide the complete settings, aggregation equations and measurement protocol, respectively.

4.2 Overall Performance

The results in Table 1 provide three observations about the effectiveness and design of sparse aggregation under large-batch training, which we summarize as follows:

Table 1: Overall performance at $128/600$ relative to Base ($32/600$). Here G denotes workers per group, c is the Capped RBI cap, and differences are in percentage points.

Method	Setting	AUC	GAUC	ΔAUC (pt)	ΔGAUC (pt)
Base	—	0.7785	0.6558	—	—
Reduce-Mean	—	0.7767	0.6498	−0.18	−0.59
RBI	Sparse LR 1×	0.7783	0.6544	−0.02	−0.13
	Sparse LR 2×	0.7784	0.6547	−0.01	−0.11
	Sparse LR 4×	0.7781	0.6540	−0.04	−0.17
Reduce-Sum	—	0.7789	0.6561	+0.04	+0.04
Capped RBI	$c = 32$	0.7788	0.6559	+0.04	+0.02
	$c = 64$	0.7789	0.6560	+0.04	+0.03
	$c = 96$	0.7788	0.6559	+0.04	+0.02
	$c = 128$	0.7797	0.6553	+0.12	−0.04
GIGA	$G = 1$	**0.7798**	0.6557	**+0.13**	−0.01
	$G = 2$	0.7790	**0.6563**	+0.05	**+0.06**
	$G = 4$	0.7789	**0.6563**	+0.04	+0.05
GIGA (state ablation)	$G = 2, Q = M^{\odot 2}$	0.7789	0.6561	+0.04	+0.04

(1) Changing the normalization or uniformly increasing the sparse learning rate is insufficient to recover large-batch quality. Relative to Base, Reduce-Mean and RBI decrease GAUC by 0.59 and 0.13 pt, respectively, and the $2\times$ and $4\times$ sparse-LR variants of RBI still fail to recover both metrics.

(2) Aggregation rules that account for cross-worker ID activity provide substantially better recovery. Reduce-Sum, Capped RBI with $c \in \{32, 64, 96\}$, and GIGA with $G \in \{2, 4\}$ all meet the joint recovery criterion. Among these configurations, GIGA with $G = 2$ achieves the best AUC–GAUC balance, exceeding Base by 0.05 pt in AUC and 0.06 pt in GAUC. The controlled $96/600$ comparison provides consistent evidence: changing only the sparse aggregation rule to GIGA with $G = 1$ improves AUC by 0.03 pt and GAUC by 0.13 pt (Figure C.2(a)).

(3) The benefit of GIGA depends on both aggregation granularity and accumulator construction. GIGA with $G = 1$ attains the highest AUC but remains 0.01 pt below Base in GAUC, whereas $G = 2$ recovers both metrics. Holding the $G = 2$ numerator fixed, replacing separate accumulation of group-wise squares with $Q = M^{\odot 2}$ reduces the AUC/GAUC gains from 0.05/0.06 pt to 0.04/0.04 pt. Together with the $G = 1$ comparison against Reduce-Sum, this result shows that the observed trade-off is not determined by the update numerator alone.

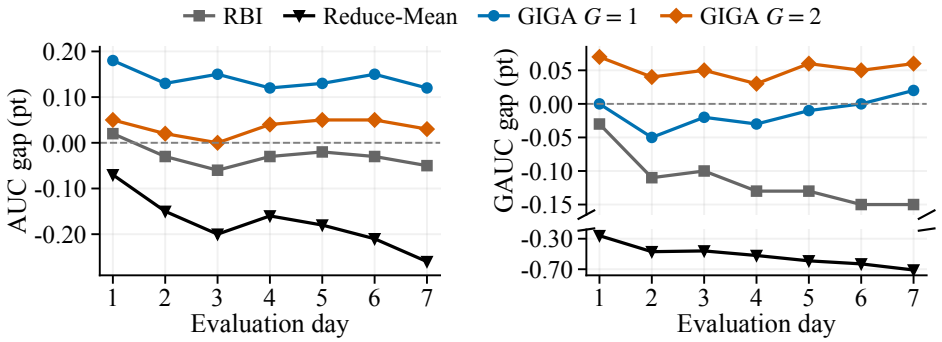


Figure 3: Seven-day AUC and GAUC gaps for GIGA $G = 1$ and $G = 2$, RBI and Reduce-Mean at $128/600$, measured against the corresponding $32/600$ baseline in percentage points. Each daily model is evaluated on the following day's examples, and dashed horizontal lines indicate zero gap.

Multi-day stability. Figure 3 shows that the aggregate differences persist across daily datasets. GIGA with $G = 2$ maintains nonnegative AUC and positive GAUC gaps on all seven days, while $G = 1$ falls below the baseline GAUC on four days despite its larger AUC gains. Both settings outperform RBI and Reduce-Mean throughout the period, confirming that intermediate grouping provides the more stable joint AUC–GAUC balance.

Table 2: End-to-end worker scaling at fixed local batch $B = 600$. Quality differences are relative to the $32/600$ reference without GIGA; scaled rows use the selected GIGA configurations. Efficiency is throughput speedup divided by $W/32$.

Training Scale			Quality Improvement		System Scaling	
W	Global batch	G	ΔAUC (pt)	ΔGAUC (pt)	Speedup	Efficiency (%)
32 (Base)	19.2K	–	—	—	1.0×	100
64	38.4K	1	+0.04	+0.06	1.6×	80
96	57.6K	2	+0.02	+0.04	2.1×	70
128	76.8K	2	+0.05	+0.06	2.9×	73
192	115.2K	2	+0.04	+0.03	3.9×	65
256	153.6K	4	+0.05	+0.03	4.8×	60
512	307.2K	8	+0.05	0.00	9.8×	61

4.3 Scalability

We next ask whether GIGA can preserve recommendation quality as the training system scales. We fix the local batch size at $B = 600$ and increase the worker count from 32 to 512, so the global batch WB grows linearly while the per-worker workload remains unchanged. At each scale, Table 2 reports the selected GIGA group size together with the corresponding end-to-end quality and throughput. The $32/600$ run without GIGA is the reference. Because the scaled rows use the selected dense-parameter settings at each worker count, this experiment evaluates the complete scaling recipe rather than the isolated effect of sparse aggregation.

Quality retention. Every selected scaled configuration meets the joint recovery criterion. Most notably, increasing the worker count and global batch by $16\times$ leaves GAUC unchanged at 512 workers while improving AUC by 0.05 pt. The intermediate scales likewise remain at or above the reference on both metrics, showing that quality does not progressively erode as more sparse updates are coalesced into each global step.

Throughput scaling. The 512-worker configuration reaches $9.8\times$ the reference throughput, or 61% end-to-end scaling efficiency over the full $16\times$ worker expansion. From 256 to 512 workers, throughput increases from $4.8\times$ to $9.8\times$, so the final doubling of workers approximately doubles throughput while retaining reference quality. These measurements include the full training stack rather than only the sparse aggregation kernel.

Overhead and group-size selection. GIGA adds approximately 3% step latency at 512 workers and at most 6.1% across the measured configurations. Estimated additional memory is below 1 GB per worker at $256/1200$ with $G = 4$. The selected group size increases with system scale, from $G = 1$ at 64 workers to $G = 8$ at 512 workers, keeping the number of groups W/G between 48 and 96. This tuning is consequential rather than cosmetic: at $512/600$, $G = 4$ and $G = 8$ recover both reference metrics, whereas $G = 2$ and $G = 16$ lose GAUC. Appendix C.1 reports the system measurements, and Appendix C.2 provides the reported group-size sensitivity results.

4.4 Generalization

To test whether the effect transfers beyond the primary recommendation workload, Table 3 reports matched comparisons on a separate search CTR workload under warm-start and from-scratch initialization. Each GIGA run is paired with a Reduce-Mean run using the same worker count, local batch size, learning rates and remaining training settings. Only the sparse aggregation rule changes. The complete configuration grids appear in Appendix E.

GIGA improves both AUC and GAUC in all six matched comparisons. Under warm-start training, the gains reach 0.05 pt in AUC and 0.09 pt in GAUC, including at 128 workers. From scratch, with $W = 32$, $G = 1$ and fixed learning rates, the gains increase monotonically with local batch size, from 0.28/0.35 pt AUC/GAUC at $B = 256$ to 1.50/1.48 pt at $B = 1024$. The complete grid also shows that $G = 1$ outperforms RBI at all three batch sizes, indicating that removing inactive-worker dilution alone does not explain the improvement.

Table 3: Matched AUC and GAUC gains of GIGA over Reduce-Mean on search CTR. Rows use the same training settings; values are percentage points (pt).

Training Configuration				Gain over Matched Reduce-Mean	
Initialization	W/B	G	Dense / sparse LR	ΔAUC (pt)	ΔGAUC (pt)
Warm-start	$32/1024$	1	$10^{-5}/10^{-3}$	$+0.02$	$+0.04$
	$32/1024$	1	$4\times10^{-5}/4\times10^{-3}$	$+0.05$	$+0.09$
	$128/1024$	4	$8\times10^{-5}/4\times10^{-3}$	$+0.05$	$+0.09$
From-scratch	$32/256$	1	$2\times10^{-5}/10^{-3}$	$+0.28$	$+0.35$
	$32/512$	1	$2\times10^{-5}/10^{-3}$	$+0.86$	$+0.81$
	$32/1024$	1	$2\times10^{-5}/10^{-3}$	$+1.50$	$+1.48$

The warm-start $128/1024$ configuration with $G = 4$ also exceeds the smaller-batch $32/256$ reference by 0.02 pt in AUC and 0.03 pt in GAUC. Because the search experiments train for one day, they establish early-training gains rather than final convergence or independently tuned batch-scaling recipes. Complementary evidence on the primary workload covers longer horizons: GIGA-based recipes recover quality at local batches 1,024 and 2,048 (Appendix C.1), and $128/600$ with $G = 2$ retains both reference metrics across seven daily evaluations (Appendix D.2). A separate one-month run yields AUC and GAUC gains of 0.05 and 0.10 pt, respectively.

5 Related Work

Frequency-aware embedding optimization. Prior work addresses skewed ID frequencies through popularity-based embedding dimensions (Ginart et al., 2021), token-dependent learning rates (Li et al., 2022), column-wise clipping (Zheng et al., 2023), and frequency-dependent regularization (Zhu et al., 2024; Li & Lyu, 2026). GIGA instead targets synchronization-induced coalescing, where a global step merges different numbers of update opportunities for head and tail IDs. It uses the current active support to couple the update numerator with the Adagrad state increment.

Sparse gradient aggregation and adaptive state. AdaBatch averages nonzero samples per coordinate to preserve sparse sample efficiency (Défossez & Bach, 2017). Adasum uses gradient geometry to approximate smaller-batch updates (Maleki et al., 2021), while GBA maintains a target global batch under asynchronous staleness (Su et al., 2022). Two-dimensional sparse parallelism globally scales the row-wise Adagrad second moment (Zhang et al., 2025). These methods modify active-support normalization, gradient combination, batch formation or global state scaling. GIGA instead operates within each synchronous step: it uses each ID's active groups and separately accumulates their squared contributions, making both update strength and state growth ID-dependent.

Distributed recommendation training systems. Large-scale systems improve sparse-embedding placement and communication through hardware–software co-design (Mudigere et al., 2022), distribution-aware sharding (Sethi et al., 2022), caching and prefetching (Agarwal et al., 2023), and unified or pipelined sparse–dense execution (Zong et al., 2025; Feng et al., 2026; Jiang et al., 2026). GIGA complements these techniques by changing the owner-side parameter and Adagrad-state update over received per-ID gradients without adding another collective.

6 Conclusion and Limitations

We identify frequency-sensitive update coalescing as a source of large-batch quality loss: frequent IDs lose more update opportunities than rare IDs, which uniform learning-rate scaling cannot rebalance. GIGA forms virtual updates from active worker groups, sums their means, and accumulates their squares separately in Adagrad. Group size controls granularity without extra collectives or per-ID state. On two industrial CTR workloads, GIGA outperforms Reduce-Mean, RBI and learning-rate scaling. With dense scaling, it preserves both reference metrics at 512 workers and a 16-fold larger batch while delivering $9.8\times$ throughput. Multi-day and search results show stability across days and tasks. Limitations include private Adagrad workloads and one-step analysis. Future work will study other optimizers, adaptive grouping, and convergence theory.

AI USE STATEMENT

AI tools were used to assist with surveying related literature, preparing figures for the experimental section, and supporting human-led code inspection. The authors are responsible for all claims, references, code implementation, experimental results and the manuscript.

REPRODUCIBILITY STATEMENT

Section 3 and Appendices A and B specify GIGA's equations, assumptions, proofs and owner-side implementation. Code is available at https://anonymous.4open.science/r/giga-review-apache2-339B. Reported configurations and evaluation protocols support independent implementation and experimental design. The industrial results rely on proprietary production data and checkpoints that cannot be released because they contain commercially sensitive company information.

REFERENCES

Saurabh Agarwal, Chengpo Yan, Ziyi Zhang, and Shivaram Venkataraman. Bagpipe: Accelerating deep recommendation model training. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613142. URL https://arxiv.org/abs/2202.12429.

Rohan Anil, Sandra Gadanho, Da Huang, Nijith Jacob, Zhuoshu Li, Dong Lin, Todd Phillips, Cristina Pop, Kevin Regan, Gil I. Shamir, Rakesh Shivanna, and Qiqi Yan. On the factory floor: ML engineering for industrial-scale ads recommendation models. In *Proceedings of the 5th Workshop on Online Recommender Systems and User Modeling*, 2022. URL https://arxiv.org/abs/2209.05310.

Alexandre Défossez and Francis Bach. AdaBatch: Efficient gradient aggregation rules for sequential and parallel stochastic gradient methods, 2017. URL https://arxiv.org/abs/1711.01761.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL https://jmlr.org/beta/papers/v12/duchi11a.html.

Chenhao Feng, Haoli Zhang, Shakhzod Ali-zade, Yanli Zhao, Liang Luo, Jennifer Cao, Lisen Deng, Siqiao Chen, Chenyu Zhao, Tristan Rice, Daniel Johnson, Min Si, Tiantu Xu, Yi Zhang, Evgenii Kolpakov, Siqi Yan, Chuanhao Zhuge, Min Ni, Bi Xue, Qunshu Zhang, and Shen Li. FreeScale: Distributed training for sequence recommendation models with minimal scaling cost. In *Proceedings of Machine Learning and Systems*, volume 8, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/f189e7580acad0fc7fd45405817ddee3-Abstract-Conference.html.

A. A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed dimension embeddings with application to memory-efficient recommendation systems. In *2021 IEEE International Symposium on Information Theory*, pp. 2786–2791, 2021. doi: 10.1109/ISIT45174.2021.9517710. URL https://arxiv.org/abs/1909.11810.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour, 2018. URL https://arxiv.org/abs/1706.02677.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://arxiv.org/abs/1705.08741.

Zhida Jiang, Zhaolong Xing, Huichao Chai, Tianxing Sun, Qiang Peng, Baopeng Yuan, Jiaxing Wang, Hua Du, Zhixin Wu, Xuemiao Li, Yikui Cao, Xinyu Liu, Yongxiang Feng, Zhen Chen, and Ke Zhang. NestPipe: Large-scale recommendation training on 1,500+ accelerators via nested pipelining, 2026. URL https://arxiv.org/abs/2604.06956.

Mang Li and Wei Lyu. Adaptive regularization for large-scale sparse feature embedding models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=QFH5mwP9oH.

Yan Li, Dhruv Choudhary, Xiaohan Wei, Baichuan Yuan, Bhargav Bhushanam, Tuo Zhao, and Guanghui Lan. Frequency-aware SGD for efficient embedding learning with provable benefits. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=OM3pXwAn5l.

Lixin Liu, Yanling Wang, Tianming Wang, Dong Guan, Jiawei Wu, Jingxu Chen, Rong Xiao, Wenxiang Zhu, and Fei Fang. Continual transfer learning for cross-domain click-through rate prediction at Taobao. In *Companion Proceedings of the ACM Web Conference 2023*, pp. 346–350, 2023. doi: 10.1145/3543873.3584625. URL https://arxiv.org/abs/2208.05728.

Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Olli Saarikivi, Tianju Xu, Vadim Eksarevskiy, Jaliya Ekanayake, and Emad Barsoum. Scaling distributed training with adaptive summation. In *Proceedings of Machine Learning and Systems*, volume 3, 2021. URL https://proceedings.mlsys.org/paper_files/paper/2021/hash/427e0e886ebf87538afdf0badb805b7f-Abstract.html.

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1222–1230, 2013. doi: 10.1145/2487575.2488200. URL https://doi.org/10.1145/2487575.2488200.

Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, et al. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022. doi: 10.1145/3470496.3533727. URL https://arxiv.org/abs/2104.05158.

Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems, 2019. URL https://arxiv.org/abs/1906.00091.

Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. RecShard: Statistical feature-based memory optimization for industry-scale neural recommendation. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 344–358, 2022. doi: 10.1145/3503222.3507777. URL https://arxiv.org/abs/2201.10095.

Wenbo Su, Yuanxing Zhang, Yufeng Cai, Kaixu Ren, Pengjie Wang, Huimin Yi, Yue Song, Jing Chen, Hongbo Deng, Jian Xu, Lin Qu, and Bo Zheng. GBA: A tuning-free approach to switch between synchronous and asynchronous training for recommendation models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/be0a8ecf8b2743a4117557c5eca0fb79-Abstract-Conference.html.

Xin Zhang, Quanyu Zhu, Liangbei Xu, Zain Huda, Wang Zhou, Jin Fang, Dennis van der Staay, Yuxi Hu, Jade Nie, Jiyan Yang, and Chunzhi Yang. Two-dimensional sparse parallelism for large scale deep learning recommendation model training, 2025. URL https://arxiv.org/abs/2508.03854.

Zangwei Zheng, Pengtai Xu, Xuan Zou, Da Tang, Zhen Li, Chenguang Xi, Peng Wu, Leqi Zou, Yijie Zhu, Ming Chen, Xiangzhuo Ding, Fuzhao Xue, Ziheng Qin, Youlong Cheng, and Yang You. CowClip: Reducing CTR prediction model training time from 12 hours to 10 minutes on 1 GPU. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11390–11398, 2023. doi:

10.1609/aaai.v37i9.26347. URL https://ojs.aaai.org/index.php/AAAI/article/view/26347. Extended discussion: arXiv:2204.06240v3.

Zirui Zhu, Yong Liu, Zangwei Zheng, Huifeng Guo, and Yang You. Helen: Optimizing CTR prediction models with frequency-wise hessian eigenvalue regularization. In *Proceedings of the ACM Web Conference 2024*, 2024. doi: 10.1145/3589334.3645463. URL https://arxiv.org/abs/2403.00798.

Hua Zong, Qingtao Zeng, Zhengxiong Zhou, Zhihua Han, Zhensong Yan, Mingjie Liu, Hechen Sun, Jiawei Liu, Yiwen Hu, Qi Wang, YiHan Xian, Wenjie Guo, Houyuan Xiang, Zhiyuan Zeng, Xiangrong Sheng, Bencheng Yan, Nan Hu, Yuheng Huang, Jinqing Lian, Ziru Xu, Yan Zhang, Ju Huang, Siran Yang, Huimin Yi, Jiamang Wang, Pengjie Wang, Han Zhu, Jian Wu, Dan Ou, Jian Xu, Haihong Tang, Yuning Jiang, Bo Zheng, and Lin Qu. RecIS: Sparse to dense, a unified training framework for recommendation models, 2025. URL https://arxiv.org/abs/2509.20883.

APPENDIX

Contents

A Additional Theoretical Analysis . . . 14

A.1 Activation Counts . . . 14

A.2 Conditional Moments under Random Grouping . . . 14

A.3 Exchangeability and Worker-Count Invariance . . . 15

A.4 Fused versus Sequential Updates . . . 15

A.5 Batch-Scaling Background . . . 16

B Implementation and Experimental Protocol . . . 16

B.1 Implementation Details . . . 16

B.2 Experimental Setup . . . 17

B.3 Baselines and Aggregation Controls . . . 17

B.4 Evaluation and Measurement Protocol . . . 18

C Scaling and Hyperparameter Studies . . . 18

C.1 Local-Batch Scaling and Training Efficiency . . . 18

C.2 Group-Size Sensitivity . . . 20

C.3 Additional Large-Scale Controls . . . 20

D Diagnostics and Multi-Day Robustness . . . 22

D.1 Activity-Bucket Diagnostics . . . 22

D.2 Multi-Day Robustness and Learning-Rate Sensitivity . . . 24

E Search CTR Generalization . . . 26

E.1 Warm-Start Configuration Grid . . . 26

E.2 From-Scratch Configuration Grid . . . 26

A Additional Theoretical Analysis

We analyze the GIGA aggregation rule under explicit probabilistic assumptions. For a single ID at one optimizer step, we suppress the ID and step indices used in Section 2: M denotes the update numerator, Q the increment to the Adagrad accumulator, H the accumulator before the step, and $H^+ = H + Q$ its value afterward. The analysis characterizes activation counts (Section A.1), conditional moments under random grouping (Section A.2), exchangeability and worker-count invariance (Section A.3), and the discrepancy between fused and sequential updates (Section A.4). Section A.5 separately summarizes the batch-scaling approximation. Together, these results provide a probabilistic and one-step basis for interpreting GIGA's behavior under changes in worker count, group size, and batch size.

A.1 Activation Counts

Assume that each sample contains ID j independently with probability $p_j \in (0, 1]$. A worker processing a local batch of B samples is active for this ID with probability $q_j = 1 - (1 - p_j)^B$. Let n_j denote the number of active workers among the W workers in a global step. Under independent, identically distributed worker batches, $n_j \sim \mathrm{Binomial}(W, q_j)$, so

$$\Pr(n_j > 0) = 1 - (1 - q_j)^W, \quad \mathbb{E}[n_j] = W q_j.$$

Since $n_j = 0$ contributes nothing to $\mathbb{E}[n_j]$, conditioning on an update event yields $\mathbb{E}[n_j \mid n_j > 0] = W q_j / [1 - (1 - q_j)^W]$. IDs with $p_j = 0$ are never updated and are excluded from this expectation.

Over a budget of N examples, with N divisible by the global batch size WB, linearity of expectation gives the expected number of active global steps as follows:

$$\mathbb{E}[\#\text{ active global steps}] = \frac{N}{WB}[1 - (1 - p_j)^{WB}].$$

For GIGA, a group of G workers is active with probability $\omega_{j,G} = 1 - (1 - p_j)^{GB}$. With $K = W/G$ groups and $N/(WB)$ global steps, the expected number of retained group contributions is $U_j^{\mathrm{virt}}(G, B) = N\omega_{j,G}/(GB)$. These expectations require only the stated marginal activation probabilities at each step. Independence across steps is unnecessary. Applying the resulting closed forms to a nonstationary production stream therefore amounts to a local stationarity approximation.

A.2 Conditional Moments under Random Grouping

Conditioning on a fixed active-worker set and a uniformly random balanced partition, we derive the conditional means of the two quantities that define a GIGA step: the update numerator M and the Adagrad-state increment Q.

A.2.1 Grouped-Gradient Mean

Fix an active-worker set S with $n = |S| > 0$, associated gradients $\{g_i\}_{i \in S}$, and active-worker mean $r = n^{-1} \sum_{i \in S} g_i$. Draw a balanced partition of the W workers uniformly at random into $K = W/G$ groups of size G, and let $\mathcal{A}$ be the set of nonempty groups, with $A = |\mathcal{A}|$. For this partition, the GIGA update numerator is $M = \sum_{k \in \mathcal{A}} \bar{g}_k$, where $\bar{g}_k$ is the mean gradient over the active workers in group k. Expanding the group means over individual workers, each active-worker gradient g_i contributes $c_i g_i$ to M, with $c_i = 1/n_{\Pi(i)}$, where $\Pi(i)$ is the group containing worker i, and $n_{\Pi(i)}$ is the number of active workers in that group. Because every partition satisfies $\sum_{i \in S} c_i = A$, symmetry gives $\mathbb{E}[c_i \mid n] = \mathbb{E}[A \mid n]/n$, and

$$\mathbb{E}[M \mid S, \{g_w\}_{w \in S}] = \mathbb{E}[A \mid n]\, r, \qquad \mathbb{E}[A \mid n] = K \left[1 - \frac{\binom{W-G}{n}}{\binom{W}{n}}\right].$$

The formula for $\mathbb{E}[A \mid n]$ follows by summing the nonempty probability over the $K = W/G$ groups, with $\binom{W-G}{n} = 0$ when $n > W - G$. At the endpoints, it reduces to n for $G = 1$ and to 1 for $G = W$. The expected numerator is therefore the active-worker mean multiplied by the expected number of nonempty groups. This result concerns M alone. The adaptive parameter step also depends on the coupled state increment Q.

A.2.2 State-Increment Mean

Retain the active set S, its size n, and the active-worker mean r from Section A.2.1. Define the coordinatewise population variance $\sigma^2 = n^{-1}\sum_{i\in S}(g_i - r)^{\odot 2}$. For any one of the $K = W/G$ groups, let X denote its number of active workers. Then X has hypergeometric mass $p_m = \binom{n}{m}\binom{W-n}{G-m}/\binom{W}{G}$. Write $a = K\Pr(X > 0)$ for the expected number of nonempty groups and $\ell = K\sum_{m>0} p_m/m$. Conditional on $X = m > 0$, the group gradient is a sample mean without replacement. For $n > 1$, its variance is $(n-m)\sigma^2/[m(n-1)]$. Summing over groups therefore gives the conditional mean:

$$\mathbb{E}[Q \mid S, g] = ar^{\odot 2} + \frac{n\ell - a}{n-1}\sigma^2.$$

When $n = 1$, the sole active worker contributes $Q = g_i^{\odot 2}$. IDs with no active worker are skipped. At the two grouping extremes, the state increment becomes $Q = \sum_{i\in S} g_i^{\odot 2}$ for $G = 1$ and $Q = r^{\odot 2}$ for $G = W$. The same expression defines a conditional-moment control that can be computed from the current gradients. Capped RBI matches neither the worker coefficients in M nor the state construction in Q. Even a method that matched $\mathbb{E}[M]$ and $\mathbb{E}[Q]$ separately would not generally match the expected adaptive step, because $M/(\sqrt{H+Q}+\epsilon)$ couples the two quantities nonlinearly.

A.3 Exchangeability and Worker-Count Invariance

The conditional calculations above average over a uniformly random balanced partition. To connect them to the implementation, condition on the training history and suppose that the worker-level activation–gradient records are exchangeable. Identically processed i.i.d. worker inputs are sufficient. A random balanced partition can then be represented by permuting the worker records before applying a fixed balanced partition. Because exchangeability leaves their joint distribution invariant under this permutation, fixed and randomly sampled balanced groups induce the same distribution over (M, Q). This equivalence is distributional rather than pointwise. Two realized partitions can produce different updates, and the argument may fail under heterogeneous sampling or worker-specific processing.

Under this equivalence, consider one coordinate of the GIGA numerator M and state increment Q. Let $q > 0$ be the worker activation probability, and let μ_B and v_B denote the conditional mean and variance of that coordinate given activation. Define the group-activation probability $\omega_G = 1-(1-q)^G$ and $\alpha_G = \sum_{m=1}^{G}\binom{G}{m}q^m(1-q)^{G-m}/m$. Conditioning on the number of active workers in each group gives the conditional moments as follows:

$$\mathbb{E}[M] = K\omega_G\mu_B, \qquad \mathbb{E}[Q] = K(\omega_G\mu_B^2 + \alpha_G v_B).$$

Over a sample budget of N examples, multiplying these per-step moments by the number of global steps, $N/(WB)$, eliminates their explicit dependence on W when G, B, and the worker-gradient distribution remain fixed. This invariance does not extend directly to changes in B: the local batch size changes both activation probabilities and the distribution induced by mean-loss normalization. The moment identities therefore do not prescribe a learning-rate scaling rule for B.

A.4 Fused versus Sequential Updates

Consider one coordinate and a fixed sequence of A group gradients $z_1, \ldots, z_A$. Let $H_0 = H$ be the pre-update Adagrad accumulator, define $H_a = H_{a-1} + z_a^2$, and write $Q = H_A - H_0 = \sum_{a=1}^{A} z_a^2$ for the total state increment. Sequential Adagrad normalizes z_a by H_a, whereas the fused GIGA update normalizes their sum by the final state H_A (Section 3.4). For $f(x) = (\sqrt{x}+\epsilon)^{-1}$, $H_0 > 0$, $\epsilon \geq 0$, and $x \geq H_0$, the derivative of f is bounded by the following:

$$|f'(x)| = \frac{1}{2\sqrt{x}(\sqrt{x}+\epsilon)^2} \leq \frac{1}{2\sqrt{H_0}(\sqrt{H_0}+\epsilon)^2}.$$

Because $\Delta_{\text{seq}} - \Delta_{\text{GIGA}} = -\eta\sum_{a=1}^{A} z_a[f(H_a) - f(H_A)]$, the mean-value theorem and triangle inequality give the following bound on the sequential and fused updates:

$$|\Delta_{\text{seq}} - \Delta_{\text{GIGA}}| \leq \frac{\eta\sum_{a=1}^{A}|z_a|(H_A - H_a)}{2\sqrt{H_0}(\sqrt{H_0}+\epsilon)^2} \leq \frac{\eta Q\sum_{a=1}^{A}|z_a|}{2\sqrt{H_0}(\sqrt{H_0}+\epsilon)^2}. \tag{A.1}$$

It does not cover a genuinely sequential optimization process, which would recompute later gradients after each intermediate parameter update.

A.5 Batch-Scaling Background

Let W denote the worker count and B the common local batch size, so the global batch is WB. Without gradient accumulation, a budget of N examples produces $T = N/(WB)$ optimizer steps. Increasing either W or B reduces T, but the consequences differ for dense parameters, which are updated at every step, and sparse IDs, which are updated only when active.

Dense-parameter heuristic. Under a frozen model and identically distributed inputs, increasing a mean-reduced batch does not scale the expected dense gradient by the inverse batch size. If the mean gradient and historical Adagrad denominator vary slowly, cumulative dense-parameter displacement is approximately proportional to $T\eta_d s_d$, where η_d is the dense learning rate and s_d is the dense gradient multiplier. For a c-fold increase in global batch, this approximation motivates

$$\eta'_d = \sqrt{c}\,\eta_{d,0}, \qquad s'_d = \sqrt{c}\,s_{d,0}.$$

The product $\eta'_d s'_d$ is then multiplied by c, offsetting the c-fold reduction in optimizer steps within this first-order approximation. Because scaling the gradient also rescales the Adagrad-state increments, this heuristic does not preserve the full optimizer trajectory.

B Implementation and Experimental Protocol

To connect the analytical quantities to the reported experiments, we specify the owner-side implementation, common training conditions, control constructions, and evaluation metrics.

B.1 Implementation Details

Algorithm 1 Owner-side GIGA aggregation and Adagrad update for active embedding IDs.

Require: Records $\mathcal{R} = \{(w, j, g_{w,j})\}$, group size G, per-ID states e_j, H_j, and η, ϵ
Require: One coalesced record per active (w, j), including active zero gradients
$\mathcal{P} \leftarrow \text{unique}\{(j, \lfloor w/G \rfloor) : (w, j, g) \in \mathcal{R}\}$
Allocate $Z_{j,k} = 0$, $n_{j,k} = 0$ only for $(j, k) \in \mathcal{P}$
for all $(w, j, g) \in \mathcal{R}$ **do**
  $k \leftarrow \lfloor w/G \rfloor$, $Z_{j,k} \leftarrow Z_{j,k} + g$, and $n_{j,k} \leftarrow n_{j,k} + 1$
end for
$M_j \leftarrow 0$, $Q_j \leftarrow 0$ for owner-local active IDs
for all $(j, k) \in \mathcal{P}$ **do**
  $\bar{g} \leftarrow Z_{j,k}/n_{j,k}$
  $M_j \leftarrow M_j + \bar{g}$ and $Q_j \leftarrow Q_j + \bar{g}^{\odot 2}$
end for
for all active IDs j **do**
  $H_j \leftarrow H_j + Q_j$
  $e_j \leftarrow e_j - \eta M_j/(\sqrt{H_j} + \epsilon)$
end for

Routing and optimizer interface. With zero-based worker ranks, the receive split sizes identify the source worker w and therefore its group $\lfloor w/G \rfloor$. The forward pass caches the group–ID mapping, and the backward pass reuses the existing sparse communication route. Before group reduction, records are coalesced to one gradient per active worker–ID pair. This ensures that the reducer count equals the number of active workers. Implementations that permit multiple lookups into a shared table must enforce the same invariant. For each active ID, Sparse Adagrad first adds the GIGA state increment Q to H and then evaluates the denominator $\sqrt{H} + \epsilon$. The accumulator is stored in the embedding dtype, FP32 by default.

Temporary memory. Let U be the number of active IDs owned by a parameter server and D the embedding width. A dense representation of all $K = W/G$ groups would require a buffer of shape $[K, U, D]$. The implementation instead materializes only the V observed group–ID pairs in a $[V, D]$ buffer, where $V \leq \min(KU, R)$ and R is the number of coalesced worker–ID records. The final

update-numerator and state-increment buffers, M and Q, each have shape $[U, D]$. Workspace can be bounded further by processing complete source groups in chunks, provided that every record for a group–ID pair is combined before the group mean is squared. Zero-valued gradients from active workers remain in the counts. Standard GIGA supplies Q separately to Adagrad. The squared-sum control instead supplies $M^{\odot 2}$.

B.2 Experimental Setup

Primary workload and system. The primary workload is a DLRM-like CTR model comprising embedding features, feature interactions, long- and short-term sequence modules, and an MLP prediction head. The main item-ID embedding has dimension 64. Collision-free hash embeddings use eviction and filtering. Training uses PyTorch/RecIS with one AMD MI308X GPU per worker, GPU-resident embeddings, and Adagrad for both dense and sparse parameters.

Training protocol. All primary-workload configurations start from the same model and optimizer checkpoint. They train incrementally on the same seven consecutive daily datasets, and each daily checkpoint is evaluated on the following day's examples. Group sizes, caps, and learning rates were selected before the reported evaluation and then held fixed across the matched comparisons. Warm-start runs use 500 steps of dense and sparse learning-rate warmup and a mean-reduced loss, with no gradient accumulation, weight decay, auxiliary regularization, gradient clipping, or update clipping. The search workload has separate warm-start and from-scratch baselines. Appendix E gives the full grids and evaluation results.

Optimizer and notation. Adagrad uses $\epsilon = 10^{-10}$. Accumulators for dense parameters and newly created sparse IDs are initialized to 0.1. We write W/B for a run with W workers and local batch size B. Unless a row is explicitly labeled as a learning-rate trial, the 128-worker aggregation comparisons use $B = 600$, dense learning rate/gradient multiplier $0.014/2$, and sparse learning rate 0.007.

B.3 Baselines and Aggregation Controls

Aggregation constructions. For one active ID, let M denote the vector placed in the numerator of the Adagrad update and Q the increment added to its coordinatewise accumulator. Table B.1 lists these two quantities for each aggregation rule. Reduce-Mean normalizes over all workers, whereas RBI normalizes over the active-worker support. Reduce-Sum and Capped RBI vary the numerator while retaining a single squared aggregate. GIGA instead retains one contribution per active group in both M and Q. The controls additionally include uniform sparse learning-rate scaling of RBI and a matched-G ablation that keeps GIGA's numerator but replaces the state increment by $M^{\odot 2}$.

Table B.1: Update numerator M and Adagrad-state increment Q for one active ID. Here S is the active-worker set, $s = \sum_{w \in S} g_w$, $n = |S|$, $r = s/n$, and $\bar{g}_k$ is the active-worker mean in group k. Reduce-Sum and Capped RBI are diagnostic controls.

Rule	Update numerator M	State increment Q
Reduce-Mean	s/W	$(s/W)^{\odot 2}$
RBI	r	$r^{\odot 2}$
Reduce-Sum	s	$s^{\odot 2}$
Capped RBI	$\min(n, c)r$	$r^{\odot 2}$
GIGA	$\sum_{k \in \mathcal{A}} \bar{g}_k$	$\sum_{k \in \mathcal{A}} \bar{g}_k^{\odot 2}$

Shared-numerator comparison. The update numerator alone does not determine an adaptive step. Let S be the active-worker set, $s = \sum_{w \in S} g_w$, $n = |S|$, and $r = s/n$. GIGA with $G = 1$, Reduce-Sum, and Capped RBI with $c \geq W$ all use $M = s$, but their state increments are $\sum_{w \in S} g_w^{\odot 2}$, $s^{\odot 2}$, and $r^{\odot 2}$, respectively. Multiplying the learning rate changes none of these state increments, so it cannot make their Adagrad trajectories equivalent. At a fixed group size G, replacing Q while retaining M isolates the state construction for the current step. Later parameters and gradients may nevertheless diverge. Section A.2.2 characterizes the conditional GIGA state increment, and Appendix B.2 gives the common training settings.

Where scaling is applied. Multiplying each input gradient by $b > 0$ before aggregation maps (M, Q) to (bM, b^2Q). Multiplying only the update numerator instead gives (bM, Q) and, for fixed current gradients without clipping, is equivalent to multiplying the learning rate by b. Capped RBI uses the latter form, $M = \min(n, c)r$ and $Q = r^{\odot 2}$: the current active-worker count determines the numerator scale, while the state remains that of RBI. It therefore requires no historical frequency estimate and can share the same aggregation path as the other controls.

Comparison scope. The sparse learning-rate sweep uses multipliers $1\times$, $2\times$, and $4\times$. Table 1 reports all tested 128-worker learning rates, caps, group sizes, and state rules, including configurations that do not meet the joint AUC–GAUC recovery criterion. Capped RBI is a practical activity-dependent scaling rule, not an implementation of the conditional-moment expression in Section A.2.2. Hyperparameters were selected in earlier experiments and then fixed for the reported evaluation. The selected configurations keep the number of groups W/G approximately between 32 and 96. The wider sweeps include values outside this range to expose the trade-off.

B.4 Evaluation and Measurement Protocol

Unless noted otherwise, quality differences on the primary workload are measured against the $32/600$ reference, whose AUC and request-level GAUC are 0.7785 and 0.6558. We report differences in percentage points (pt), with 1 pt corresponding to 0.01 in the underlying metric.

B.4.1 Reference Metric Definitions

For an evaluation set containing n_+ positive and n_- negative examples, AUC assigns half credit to tied scores and is computed as follows:

$$\mathrm{AUC} = \frac{\sum_{i:y_i=1} \sum_{j:y_j=0} [\mathbb{I}(s_i > s_j) + \frac{1}{2}\mathbb{I}(s_i = s_j)]}{n_+ n_-}.$$

For the set $\mathcal{V}$ of eligible requests containing both positive and negative examples, sample-weighted GAUC is defined as the mean of AUC over each request:

$$\mathrm{GAUC} = \frac{\sum_{u \in \mathcal{V}} n_u \, \mathrm{AUC}_u}{\sum_{u \in \mathcal{V}} n_u}.$$

Here u indexes requests containing both labels, and n_u is the number of evaluated examples in request u. Requests with only positive or only negative labels are omitted from both the numerator and denominator. GAUC is computed after assembling complete requests, so neither batch nor worker boundaries split a request. The same procedure is used at every training scale.

B.4.2 Repeat-Run Variability

Across five matched daily evaluations, two independent $32/600$ baseline runs differ by 0.010 pt in AUC and 0.008 pt in GAUC on average. The maximum absolute difference is 0.010 pt for either metric. Historical repeated runs typically differ by less than 10^{-4} in the underlying metric. These measurements quantify run-to-run variability but are not used as a formal significance threshold.

C Scaling and Hyperparameter Studies

The main-text scaling results are expanded along three axes: local batch size, group size, and worker count. Throughout this section, W/B denotes W workers with local batch size B.

C.1 Local-Batch Scaling and Training Efficiency

Table C.1 extends the worker-scaling study in Section 4.3 to the $96/1024$ setting. Its rows form a cumulative recipe rather than a single-factor ablation: GIGA uses $G = 2$, and the first two additions also modify the dense-parameter settings. Here "**loss scale**" denotes a fixed multiplier of 1.707, not automatic mixed-precision overflow scaling. The final row changes only the normalization batch: Virtual BN computes statistics over chunks of 512 examples, following Ghost BN (Hoffer et al., 2017).

Table C.1: Cumulative training-recipe changes at $96/1024$. AUC and GAUC differences are measured against the 32-worker reference and reported in percentage points (pt).

Stage	ΔAUC (pt)	ΔGAUC (pt)
Direct expansion	-0.02	-0.29
+ GIGA and dense adjustments	$+0.08$	-0.01
+ Loss scale and dense adjustments	$+0.08$	$+0.02$
+ Virtual BN	+0.00	$+0.04$

A separate $96/1024$ configuration without loss scaling uses $G = 2$, dense learning rate/gradient scale $0.016/2.260$, and sparse learning rate 0.012. Its AUC and GAUC gaps are $+0.01$ and $+0.04$ pt. At $96/2048$, the selected $G = 4$ recipe uses dense learning rate/gradient scale $0.022/3.2$, sparse learning rate 0.022, sparse post-scale 1, and normalization chunks of 512. Its gaps are 0.00/0.01 pt. These are combined scaling recipes: they jointly modify sparse aggregation, the effect of local mean-loss normalization, and normalization statistics. Virtual BN shares parameters across chunks and does not introduce additional optimizer steps.

Efficiency under worker scaling. The speedups in Table 2 are ratios of average end-to-end training throughput, measured in examples per second, relative to the 32-worker run. The 512-worker configuration reaches $9.8\times$ the reference throughput and uses the same $G = 8$ setting as the corresponding quality measurement. Relative to the underlying training step, GIGA adds approximately 3% latency at 512 workers and no more than 6.1% in the measured configurations. At $256/1200$ with $G = 4$, its estimated additional memory is below 1 GB per worker.

End-to-end elapsed time. Figure C.1 measures wall-clock progress through the same seven-partition window. Unlike the throughput measurements, these times include both training and the prediction pass used for evaluation. Total elapsed time decreases from 52.23 hours at 32 workers to 16.36, 11.56, 8.39, and 5.04 hours at 96, 128, 256, and 512 workers, respectively.

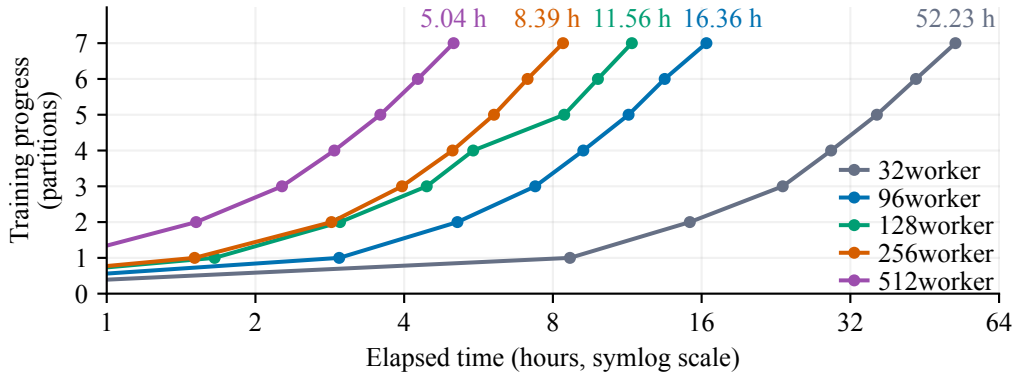


Figure C.1: End-to-end progress across worker scales. Elapsed time includes training on partition T and prediction on partition $T + 1$.

To distinguish the effect of sparse aggregation from uniform learning-rate adjustment, Figure C.2 compares matched RBI and GIGA configurations, while Table C.2 reports learning-rate-only controls. At $96/600$, replacing RBI by GIGA with $G = 1$ while holding the remaining settings fixed increases AUC by 0.03 pt and GAUC by 0.13 pt. In the $512/600$ sweep, the dense learning rate/gradient multiplier remains $0.028/4$, and the sparse learning rate/gradient multiplier remains $0.007/1$. Under these fixed settings, $G = 4$ and $G = 8$ recover both reference metrics, whereas $G = 2$ increases AUC but leaves GAUC below the reference.

With the dense learning-rate multiplier fixed at $\sqrt{8}$, increasing the sparse multiplier from 1 to $\sqrt{8}$ narrows the GAUC deficit from 0.24 to 0.15 pt, but neither setting reaches the reference. Scaling both rates linearly by 8 degrades both metrics further. The daily curves in Figure D.5 reinforce this

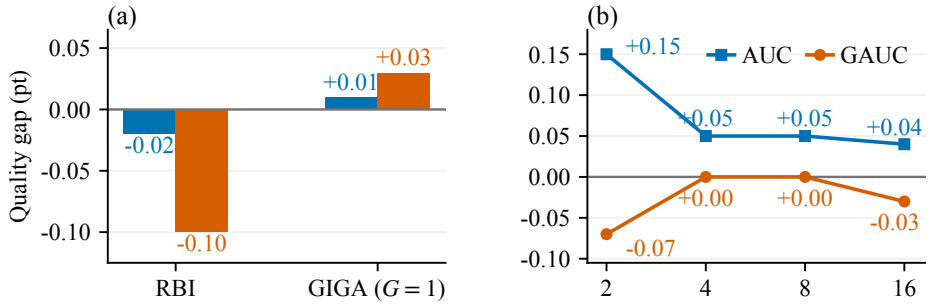


Figure C.2: AUC and GAUC gaps relative to the 32-worker baseline for a matched RBI–GIGA comparison at $96/600$ and a GIGA group-size sweep at $512/600$.

Table C.2: Learning-rate scaling without GIGA at 256 workers. AUC and GAUC differences are relative to the 32-worker reference and reported in percentage points. The rows use unchanged, square-root, and linear multipliers of the baseline rates.

Dense LR multiplier	Sparse LR multiplier	ΔAUC (pt)	ΔGAUC (pt)
$\sqrt{8}\times$	$1\times$	-0.04	-0.24
$\sqrt{8}\times$	$\sqrt{8}\times$	-0.01	-0.15
$8\times$	$8\times$	-0.06	-0.29

aggregate result: neither the $2\times$ nor the $4\times$ RBI setting recovers baseline GAUC on any day, whereas GIGA with $G = 2$ remains at or above both baseline metrics throughout the seven-day window. Together with the matched comparison in Figure C.2(a), these results show that uniform learning-rate scaling does not reproduce the recovery obtained by changing the sparse aggregation rule.

C.2 Group-Size Sensitivity

Table C.3 reports group-size sensitivity at fixed local batch size over the configurations used in the scaling study. The dense learning rate η_d and gradient multiplier s_d are shown separately so that group-size effects can be compared within each worker-count block.

The sweep shows a consistent AUC–GAUC trade-off. At 128 and 192 workers, $G = 1$ gives the largest AUC gain but lower GAUC than intermediate group sizes. At 256 and 512 workers, recovering both reference metrics requires larger groups, consistent with Section 3.3. Increasing G beyond the selected value is not uniformly beneficial: at 128 workers, $G = 8$ and $G = 16$ reduce the gains relative to $G = 2$, and at 512 workers, $G = 16$ again loses reference GAUC. The selected group sizes $G = (2, 2, 2, 4, 8)$ for $W = (96, 128, 192, 256, 512)$ correspond to $W/G = (48, 64, 96, 64, 64)$. Thus, the chosen configurations keep the number of groups within a relatively narrow range while allowing group size to grow with worker count.

C.3 Additional Large-Scale Controls

The group-size sweep identifies operating points that recover both metrics. The controls below compare those points with alternative numerators and state constructions at 256 and 512 workers.

C.3.1 256-Worker Controls

The 256-worker controls use local batch 600 and dense learning rate/gradient multiplier $0.020/2.83$. All quality gaps are relative to the common $32/600$ baseline.

Activity-dependent effect of group size. Figure C.3 resolves the update numerator and parameter-update norms by active-worker count, allowing the effects of group size and state construction to be compared within activity ranges.

Table C.3: Group-size sensitivity at local batch 600. AUC and GAUC gaps are measured against the $32/600$ reference. The columns η_d and s_d denote the dense learning rate and dense gradient multiplier, respectively. † marks the two 96-worker configurations that modify the default dense setting, and shaded rows denote the selected configuration at each worker scale.

W	**Dense setting**		G	**Gap vs.** $32/600$ **(pt)**	
	η_d	s_d		ΔAUC	ΔGAUC
96	0.007	1.00	1	+0.01	+0.03
96	0.012	1.732	1†	+0.00	+0.04
96	**0.010**	**1.41**	**2**†	+0.02	+0.04
96	0.007	1.00	4	+0.02	+0.03
96	0.007	1.00	8	+0.01	+0.02
96	0.007	1.00	16	+0.01	+0.00
128	0.014	2.00	1	+0.13	−0.01
128	**0.014**	**2.00**	**2**	+0.05	+0.06
128	0.014	2.00	4	+0.04	+0.05
128	0.014	2.00	8	+0.03	+0.02
128	0.014	2.00	16	+0.02	+0.00
192	0.017	2.45	1	+0.14	−0.03
192	**0.017**	**2.45**	**2**	+0.04	+0.03
192	0.017	2.45	4	+0.03	+0.03
192	0.017	2.45	8	+0.02	+0.02
192	0.017	2.45	16	+0.02	+0.00
256	0.020	2.83	1	+0.14	−0.04
256	0.020	2.83	2	+0.15	−0.02
256	**0.020**	**2.83**	**4**	+0.05	+0.03
256	0.020	2.83	8	+0.04	+0.03
256	0.020	2.83	16	+0.04	+0.02
512	0.028	4.00	1	+0.14	−0.06
512	0.028	4.00	2	+0.15	−0.07
512	0.028	4.00	4	+0.05	+0.00
512	**0.028**	**4.00**	**8**	+0.05	+0.00
512	0.028	4.00	16	+0.04	−0.03

Table C.4: Aggregation controls at $256/600$. AUC and GAUC gaps are relative to the $32/600$ reference and reported in percentage points. Bold marks the selected GIGA configuration, $G = 4$.

Method	ΔAUC (pt)	ΔGAUC (pt)
RBI	-0.04	-0.23
GIGA $G = 1$	+0.14	-0.04
GIGA $G = 2$	+0.15	-0.02
GIGA $G = 4$	**+0.05**	**+0.03**
GIGA $G = 8$	+0.04	+0.03
GIGA $G = 16$	+0.04	+0.02
Capped RBI $c = 64$	+0.05	+0.00
Reduce-Sum	+0.05	+0.01
GIGA $G = 4$, $Q = M^{\odot 2}$	+0.05	+0.00

The effect of grouping is concentrated among highly active IDs. In the highest-activity bucket at step 5,000, increasing G from 1 to 4 lowers the mean update norm from 3.90×10^{-5} to 9.94×10^{-6} under the standard separate-square state, a factor of 3.93. Under $Q = M^{\odot 2}$, it falls from 4.33×10^{-5} to 1.15×10^{-5}, a factor of 3.77. Capped RBI with $c = 64$ yields 1.06×10^{-5} in the same bucket. These are aggregate bucket statistics from separate runs, not paired updates evaluated on identical gradients, and bucket membership may differ across methods. They establish the activity-dependent reduction in update magnitude but do not by themselves determine the quality trade-off in Figure D.4.

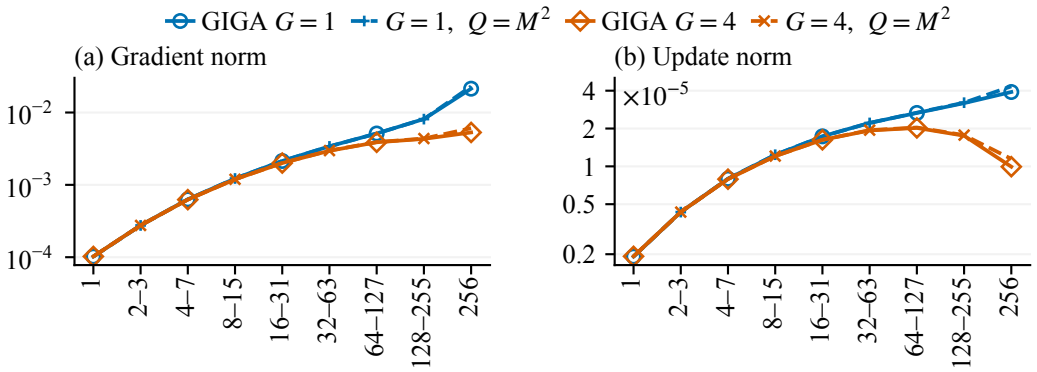


Figure C.3: Mean numerator norm $\|M_j\|_2$ and parameter-update norm $\|\Delta e_j\|_2$ by active-worker-count bucket at step 5,000 of the $256/600$ runs. Colors distinguish $G = 1$ from $G = 4$. Solid curves use the standard separate-square state, and dashed curves use $Q = M^{\odot 2}$. Markers alternate across buckets to make overlapping curves visible. Lines include every bucket.

For standard GIGA with $G = 1$, the highest-activity bucket has coherence 0.692 at step 5,000, while the recorded trajectory fluctuates around one between steps 1,000 and 10,000. The snapshot therefore does not support a persistent cancellation claim. More generally, aggregate coherence near one does not imply coordinatewise equality between the separate-square state and $M^{\odot 2}$.

C.3.2 512-Worker Controls

The 512-worker controls use local batch 600, dense learning rate/gradient multiplier $0.028/4.00$, and sparse learning rate 0.007. At 512 workers, RBI is 0.34 pt below the reference GAUC. Capped RBI,

Table C.5: Aggregation controls at $512/600$. AUC and GAUC gaps are relative to the $32/600$ reference and reported in percentage points. Bold marks the selected GIGA configuration, $G = 8$.

Method	ΔAUC (pt)	ΔGAUC (pt)
RBI	-0.07	-0.34
GIGA $G = 1$	+0.14	-0.06
GIGA $G = 8$	**+0.05**	**+0.00**
Capped RBI $c = 64$	+0.05	+0.00
Reduce-Sum	+0.05	+0.00

Reduce-Sum, and GIGA with $G = 8$ all recover the reference GAUC at the reported precision and produce the same 0.05-pt AUC gain. GIGA with $G = 1$ gives a larger AUC gain but remains 0.06 pt below the reference GAUC, making $G = 8$ the more balanced operating point.

D Diagnostics and Multi-Day Robustness

Aggregate AUC and GAUC values do not reveal how an aggregation rule redistributes updates across IDs. We therefore pair activity-bucket diagnostics with daily quality curves to connect one-step behavior to the observed differences across methods.

D.1 Activity-Bucket Diagnostics

Figure D.1 reports one-step diagnostics for the representative item-ID table at step 5,000 of the $128/600$ run. For ID j, M_j denotes the GIGA update numerator, Q_j the Adagrad-state increment, and H_j the pre-update accumulator. The resulting embedding update is $\Delta e_j = -\eta M_j/(\sqrt{H_j + Q_j} + \epsilon)$. IDs are grouped by their number of distinct active workers: 1, 2–3, 4–7, 8–15, 16–31, 32–63, 64–127, and 128. The reported gradient and update norms are bucket averages of $\|M_j\|_2$ and $\|\Delta e_j\|_2$.

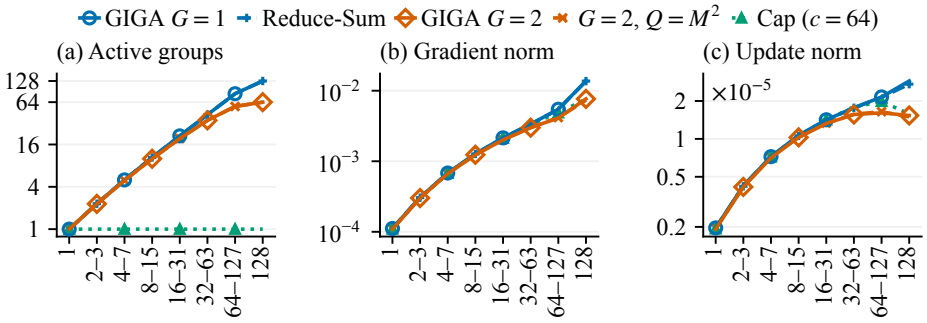


Figure D.1: Activity-bucket analysis at step 5,000 of the 128/600 runs: (a) mean active-group count, (b) mean numerator norm $\|M_j\|_2$, and (c) mean parameter-update norm $\|\Delta e_j\|_2$. Buckets are defined by the number of distinct active workers. Solid and dashed curves use the standard separate-square state and $Q = M^{\odot 2}$, respectively. The dotted "Cap" curve is Capped RBI with $c = 64$. Alternating markers distinguish overlapping curves, while each line retains all bucket values.

The curves separate mainly in the high-activity buckets. Low-activity IDs have nearly identical gradient and update norms under $G = 1$ and $G = 2$, whereas highly active IDs occupy fewer groups and receive smaller updates as G increases. In the 128-worker bucket, the mean update norm under standard GIGA is $1.88\times$ larger at $G = 1$ than at $G = 2$. At a fixed G, the separate-square state $Q_j = \sum_k \bar{g}_{j,k}^{\odot 2}$ and the ablated state $Q_j = M_j^{\odot 2}$ produce similar norm profiles in this snapshot despite different quality outcomes. The multi-day ablation therefore provides the stronger basis for selecting the state rule. Capped RBI approaches the $G = 2$ update norm in the highest-activity bucket but differs in the intermediate buckets. Because these statistics come from separate runs at one optimizer step, the bucket membership need not be identical across methods.

Effect of the activity cap. Capped RBI uses $M = \min(n, c)r$, so c limits the activity-dependent multiplier applied to the active-worker mean r. Figure D.2 traces the resulting scale, saturation frequency, and update norm across activity buckets.

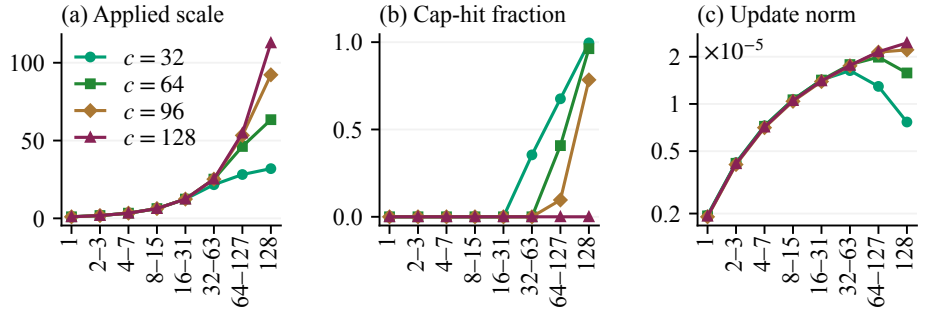


Figure D.2: Capped-RBI analysis at 128/600: applied multiplier $\min(n, c)$, fraction of reducer events with $n > c$, and mean parameter-update norm $\|\Delta e_j\|_2$, grouped by active-worker-count bucket.

As activity increases, the applied multiplier saturates at c and the fraction of capped reducer events rises. Smaller values of c therefore reduce the update norm earlier, while leaving low-activity IDs largely unchanged. The cap-hit fraction records truncation of the activity multiplier. It is not a gradient-clipping rate. Because the statistics are accumulated over reducer applications, repeated lookup paths can make the event distribution differ from the distribution over unique IDs. The corresponding AUC–GAUC trade-off is reported in Section 4.2.

Aggregate cross-term diagnostic. For ID j, let $\mathcal{A}_j$ be its active-group set, $\bar{g}_{j,k}$ the mean gradient from group k, and $M_j = \sum_{k \in \mathcal{A}_j} \bar{g}_{j,k}$. We summarize the aggregate cross terms by the event-coherence ratio as the following:

$$C = \frac{\sum_j \|M_j\|_2^2}{\sum_j \sum_{k \in \mathcal{A}_j} \|\bar{g}_{j,k}\|_2^2}.$$

Across the aligned standard-GIGA snapshots, C ranges from approximately 1.002 to 1.034, so the net cross-term contribution is small after aggregation over IDs. This aggregate relation does not imply coordinatewise equality of the state increments. We therefore use coherence only to summarize the snapshot geometry and rely on the quality ablations to compare state rules. For the ablated rule $Q = M^{\odot 2}$, the analogous ratio computed from the optimizer inputs is exactly one by construction and is not a comparable empirical coherence statistic.

D.2 Multi-Day Robustness and Learning-Rate Sensitivity

The preceding diagnostics describe individual optimizer steps. The daily evaluations below test whether the corresponding quality differences persist across data partitions, state constructions, worker scales, and sparse learning rates.

D.2.1 128-Worker Aggregation and State Ablations

Every daily checkpoint is evaluated on the next day's examples, with the same evaluation set shared by all configurations for that day. Figure 3 reports the main aggregation comparison. Figure D.3 separates the state-rule and capped-RBI effects discussed in Section 4.2. Daily gaps are computed from the unrounded metrics relative to the matched 32-worker baseline and then converted to percentage points.

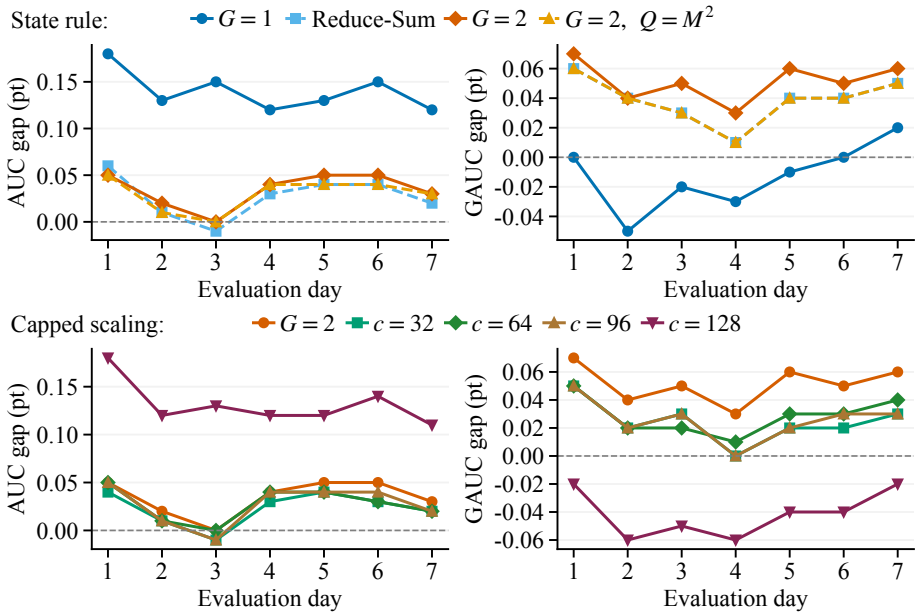


Figure D.3: Daily AUC and GAUC gaps at 128/600, relative to the matched 32/600 baseline and reported in percentage points. In the state-rule panels, solid curves use the standard separate-square state and dashed curves use $Q = M^{\odot 2}$. Unqualified $G = 1$ and $G = 2$ labels denote standard GIGA. In the capped-RBI panels, c is the activity cap. Each legend applies to both panels in its row.

At $G = 1$, replacing the standard state increment $Q = \sum_k \bar{g}_k^{\odot 2}$ by $Q = M^{\odot 2}$ raises GAUC but lowers AUC. At $G = 2$, the standard separate-square state gives higher GAUC on six days and ties on the remaining day. In the capped-RBI comparison, GIGA with $G = 2$ exceeds caps 32–96 in GAUC on every day while matching or improving their AUC. The $c = 128$ setting shifts the trade-off toward AUC but remains below the baseline GAUC throughout.

D.2.2 Worker-Scale and Learning-Rate Comparisons

Figure D.4 extends the aggregation and state comparisons to 256 workers, and Figure D.5 resolves the 128-worker learning-rate sweep by day. Together they distinguish changes in aggregation granularity from uniform rescaling of the sparse update.

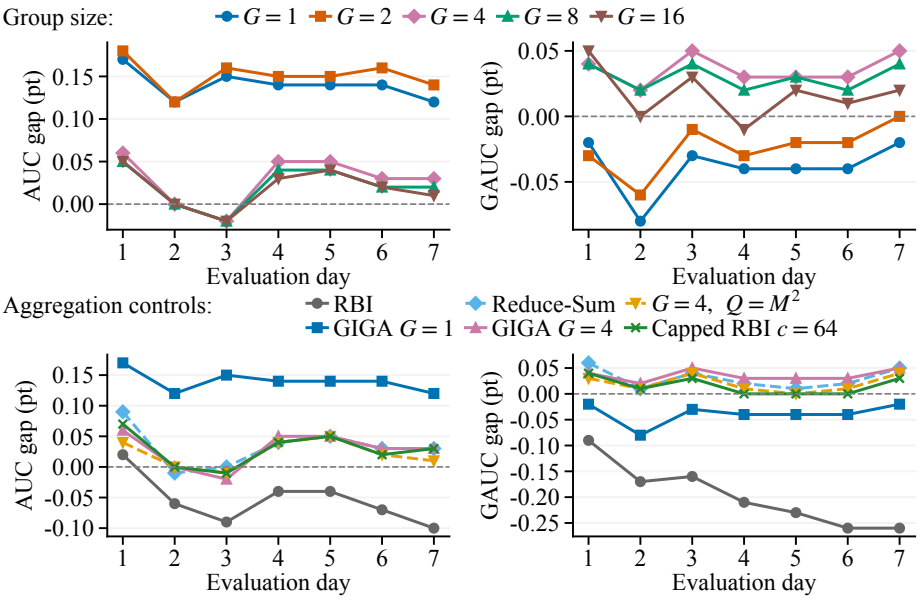


Figure D.4: Seven-day AUC and GAUC gaps at $256/600$, relative to the matched $32/600$ baseline and reported in percentage points. The rows compare group size, state construction, and alternative aggregation rules. Unqualified G labels denote standard GIGA. Colors follow each row's shared legend, and horizontal dashed lines mark zero gap.

At 256 workers, $G = 1$ improves AUC but remains below the baseline GAUC on all seven days. Intermediate group sizes yield a more balanced pair of metrics. Standard GIGA with $G = 1$ nevertheless exceeds RBI in both metrics throughout. The state-rule comparison repeats the 128-worker pattern: $Q = M^{\odot 2}$ trades AUC for GAUC at $G = 1$, whereas the standard separate-square state improves GAUC at $G = 4$ on every day. The standard $G = 4$ configuration has a small negative AUC gap on Day 3, so it does not satisfy the joint recovery criterion uniformly across days.

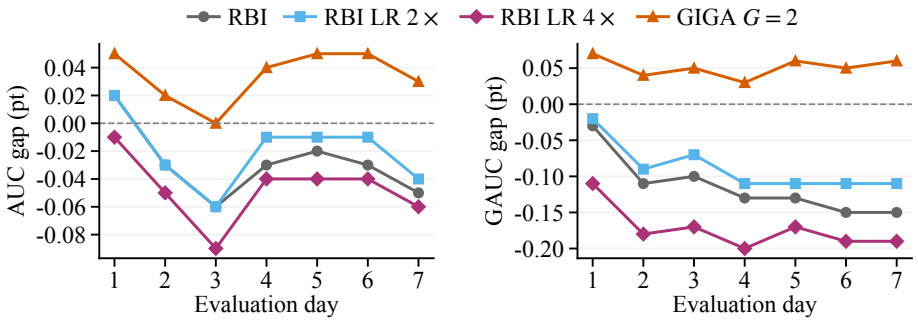


Figure D.5: Daily AUC and GAUC gaps at $128/600$ for RBI with sparse-learning-rate multipliers $1\times$, $2\times$, and $4\times$, together with GIGA $G = 2$. Gaps are relative to the matched $32/600$ baseline and reported in percentage points. Horizontal dashed lines mark zero.

Doubling the sparse learning rate narrows RBI's GAUC deficit on every day but never reaches the baseline. A $4\times$ multiplier produces lower GAUC and no higher AUC than the default RBI setting throughout the seven-day window. Under the same evaluation protocol, GIGA with $G = 2$ remains at or above both baseline metrics, indicating that uniform learning-rate scaling does not reproduce the recovery obtained by changing the aggregation rule.

Table E.1: Warm-start search CTR grid. AUC and GAUC differences are relative to the $32/256$ Reduce-Mean reference in the first row and reported in percentage points. Bold marks the selected configurations. At 128 workers, $G = 2$ and $G = 4$ tie at the reported precision.

W/B	G	d/s	s_d	ΔAUC (pt)	ΔGAUC (pt)
32/256 (baseline)	–	1/1	1	+0.00	+0.00
32/1024	–	1/1	1	-0.01	-0.02
32/1024	1	1/1	1	+0.01	+0.02
32/1024	1	4/1	1	+0.01	+0.02
32/1024	**1**	**4/4**	**1**	**+0.04**	**+0.07**
32/1024	1	2/2	1	+0.02	+0.04
32/1024	4	1/1	1	+0.00	+0.01
32/1024	–	4/4	1	-0.01	-0.02
128/1024	1	8/4	2	+0.01	+0.03
128/1024	**2**	**8/4**	**2**	**+0.02**	**+0.03**
128/1024	**4**	**8/4**	**2**	**+0.02**	**+0.03**
128/1024	–	8/4	2	-0.03	-0.06

E Search CTR Generalization

Finally, we evaluate whether the aggregation effect transfers to a separate search CTR workload under both warm-start and from-scratch training. Each run uses one day of production examples, and non-GIGA configurations use global Reduce-Mean. Here W/B denotes W workers with local batch size B, G is the number of workers per GIGA group, and GAUC is computed at the request level.

E.1 Warm-Start Configuration Grid

The warm-start grid uses the $32/256$ Reduce-Mean run as its common reference, with AUC 0.9019 and GAUC 0.8742. In Table E.1, d/s gives the dense and sparse learning-rate multipliers relative to $10^{-5}/10^{-3}$, and s_d is the dense gradient multiplier.

At 128 workers, $G \in \{1, 2, 4\}$ gives the same request GAUC, 0.8745, while $G = 2$ and $G = 4$ also tie at AUC 0.9021. Thus, several group sizes recover the degradation of the corresponding no-GIGA configuration. Table 3 in the main text reports gains against each configuration's matched Reduce-Mean run, whereas Table E.1 reports all gaps against the common $32/256$ reference. For example, at $128/1024$ and $G = 4$, subtracting the no-GIGA gaps $(-0.03, -0.06)$ from the GIGA gaps $(+0.02, +0.03)$ yields matched AUC/GAUC gains of $+0.05/+0.09$ pt.

E.2 From-Scratch Configuration Grid

All from-scratch configurations use dense learning rate 2×10^{-5}, sparse learning rate 10^{-3}, weight decay 10^{-6}, and no gradient clipping. Holding these settings fixed across local batch sizes isolates the change in sparse aggregation within each same-batch comparison.

Initialization. The from-scratch runs initialize both the model and optimizer. Sparse embedding parameters start at zero, with their Adagrad accumulators initialized to 0.1. Dense adaptive fully connected layers use Xavier-uniform weights with gain 1. Other linear layers draw weights and biases from $\mathcal{U}(-1/\sqrt{d_{\text{in}}}, 1/\sqrt{d_{\text{in}}})$, where d_{in} is the input width. Normalization scales start at 1 and biases at 0. Both sparse and standard dense Adagrad use initial accumulators of 0.1 and $\epsilon = 10^{-10}$.

For each local batch size, GIGA with $G = 1$ improves both AUC and GAUC over the matched Reduce-Mean baseline. Compared with $G = 32$, which is equivalent to RBI for 32 workers, the $G = 1$ gains are 0.28/0.24, 0.72/0.69, and 1.20/1.00 pt in AUC/GAUC at local batches 256, 512, and 1,024, respectively. Active-support normalization alone therefore does not account for the improvement. The learning rates are fixed across these comparisons, so each batch-size block isolates the aggregation rule rather than an independently tuned scaling recipe. The deltas are normalized to a

Table E.2: From-scratch search CTR grid with 32 workers and fixed learning rates. Each local-batch block begins with its matched Reduce-Mean baseline. AUC and GAUC differences are computed from the displayed absolute metrics and reported in percentage points relative to that baseline.

Local batch	G	AUC	GAUC	ΔAUC (pt)	ΔGAUC (pt)
256 (baseline)	–	0.8619	0.8187	+0.00	+0.00
256	1	0.8647	0.8222	+0.28	+0.35
256	4	0.8637	0.8212	+0.18	+0.25
256	8	0.8635	0.8212	+0.16	+0.25
256	32	0.8619	0.8198	+0.00	+0.11
512 (baseline)	–	0.8542	0.8118	+0.00	+0.00
512	1	0.8628	0.8199	+0.86	+0.81
512	8	0.8605	0.8183	+0.63	+0.65
512	32	0.8556	0.8130	+0.14	+0.12
1024 (baseline)	–	0.8429	0.8009	+0.00	+0.00
1024	1	0.8579	0.8157	+1.50	+1.48
1024	4	0.8561	0.8131	+1.32	+1.22
1024	8	0.8571	0.8148	+1.42	+1.39
1024	32	0.8459	0.8057	+0.30	+0.48

different Reduce-Mean baseline in each block and should not be used to compare absolute quality across batch sizes.