

A Appendix / supplemental material

B Modifying AdamW for subspace networks

Our approach can be framed as a constrained optimization problem in which the projection matrices must remain within the subspace \mathcal{S} . To ensure they stay in this feasible set throughout training, it requires projecting them onto \mathcal{S} at each training iteration. We propose a modified version of the AdamW optimizer that preserves the row space of $\mathbf{W}_{p_2}^l$ in \mathcal{S} . That means, once initialized within \mathcal{S} , the modified optimizer ensures that the $\mathbf{W}_{p_2}^l$ no longer require iterative projection steps, guaranteeing they remain in \mathcal{S} without incurring any approximation error. Note that however, we still need to project $\mathbf{W}_{p_1}^l$ onto \mathcal{S} due to the nonlinearity of activations as discussed in Appendix. C.

AdamW defines the momentum term as $\mathbf{M}_t = \beta_1 \mathbf{M}_{t-1} + (1 - \beta_1) \nabla_L(\mathbf{W}_{p_2}(t))$ and the second order momentum $\mathbf{V}_t = \beta_2 \mathbf{V}_{t-1} + (1 - \beta_2)(\nabla_L(\mathbf{W}_{p_2}(t)))^2$. Then, the weight update is defined as

$$\mathbf{W}_{p_2}(t+1) = \mathbf{W}_{p_2}(t) - \eta \alpha_t \hat{\mathbf{M}}_t - \lambda \mathbf{W}_{p_2}(t), \quad (12)$$

where $\alpha_t = \frac{1}{\sqrt{\hat{\mathbf{V}}_t + \epsilon}}$, $\hat{\mathbf{V}}_t = \frac{\mathbf{V}_t}{\beta_2 + \epsilon}$, $\hat{\mathbf{M}}_t = \frac{\mathbf{M}_t}{\beta_1 + \epsilon}$, and η is the fixed learning rate. λ is a hyperparameter controlling the adaptive weight decay. Note that in the second term on the right hand side in Eq. 12, α_t is not a constant scaling factor across coordinates. Thus, it changes the direction of the rows of $\hat{\mathbf{M}}_t$. This causes the $\mathbf{W}_{p_2}(t+1)$ to drift from \mathcal{S} . Therefore, we make α_t constant row-wise so that the resulting update $\eta \alpha_t \hat{\mathbf{M}}_t - \lambda \mathbf{W}_{p_2}(t)$ is closed within \mathcal{S} . Specifically, we alter $\hat{\mathbf{V}}_t$ as below. We first take

$$\mu_{\text{row}}(\hat{\mathbf{V}}_t) = \frac{1}{m} \sum_{i=1}^m \hat{\mathbf{V}}_t(:, i) \in \mathbb{R}^{n \times 1}. \quad (13)$$

where $\mu_{\text{row}}(\cdot)$ is the row-wise mean. Then,

$$\tilde{\mathbf{V}}_t = \mathbf{1}_m \cdot \mu_{\text{row}}(\hat{\mathbf{V}}_t), \quad (14)$$

where $\mathbf{1}_m \in \mathbb{R}^{1 \times m}$ is a column vector of ones, and $\tilde{\mathbf{V}} \in \mathbb{R}^{n \times m}$ has the same dimensions as $\hat{\mathbf{V}}_t$. Then we substitute $\hat{\mathbf{V}}_t$ with $\tilde{\mathbf{V}}$ in Eq. 12. We only apply the above modification to \mathbf{W}_{p_2} and keep the usual updates as it is for other weights.

C Gradient compression in backpropagation

In this section, we show that compressing the gradients by projecting them on to \mathcal{S} does not induce any approximation error.

Let $\nabla_l(\mathbf{X}^{l+1})$ denote the gradient of the loss with respect to \mathbf{X}^{l+1} , the output of layer l . The gradient with respect to $\mathbf{W}_{p_2}^l$ is then

$$\nabla_l(\mathbf{W}_{p_2}^l) = (\mathbf{X}_{\text{hidden}}^l)^\top \nabla_l(\mathbf{X}^{l+1}). \quad (15)$$

The residual gradient from the skip connection is given by

$$(\nabla_l(\mathbf{X}_{\text{attn}}^l))^{\text{residual}} = \nabla_l(\mathbf{X}^{l+1}). \quad (16)$$

Hence, the update rule for $\mathbf{W}_{p_2}^l$ becomes

$$\mathbf{W}_{p_2}^l(t+1) = \mathbf{W}_{p_2}^l(t) - \gamma (\mathbf{X}_{\text{hidden}}^l)^\top \nabla_l(\mathbf{X}^{l+1}), \quad (17)$$

where γ is the learning rate, and t denotes the training timestep.

If we project $\nabla_l(\mathbf{X}^{l+1})$ onto $\text{Col}(\mathbf{U}_k) = \mathcal{S}$, the update can be written as

$$\mathbf{W}_{p_2}^l(t+1) = \mathbf{W}_{p_2}^l(t) - \gamma (\mathbf{X}_{\text{hidden}}^l)^\top (\nabla_l(\mathbf{X}^{l+1}) \mathbf{U}_k \mathbf{U}_k^\top). \quad (18)$$

If γ acts as a row-wise constant, then it is straightforward to see that if $\text{Row}(\mathbf{W}_{p_2}^l(t)) \subseteq \mathcal{S}$, it follows that $\text{Row}(\mathbf{W}_{p_2}^l(t+1)) \subseteq \mathcal{S}$ as well, because vector spaces are closed under linear operations.

Recall that we modify AdamW so that its adaptive learning rate is constant across each row (see Section B), ensuring this property holds. **Therefore, by induction, if $\text{Row}(\mathbf{W}_{p_2}^l(0)) \subseteq \mathcal{S}$ at initialization, then $\text{Row}(\mathbf{W}_{p_2}^l(t)) \subseteq \mathcal{S}$ for all subsequent updates when the incoming gradients $\nabla_l(\mathbf{X}^{l+1})$ are projected onto \mathcal{S} , removing the need for iterative projection of $\mathbf{W}_{p_2}^l$ on to \mathcal{S} .**

Continuing backpropagation, the gradient with respect to $\mathbf{X}_{\text{hidden}}^l$ is

$$\nabla_l(\mathbf{X}_{\text{hidden}}^l) = \nabla_l(\mathbf{X}^{l+1}) \mathbf{U}_k \mathbf{U}_k^\top (\mathbf{W}_{p_2}^l)^\top. \quad (19)$$

Since $\text{Row}(\mathbf{W}_{p_2}^l) = \mathcal{S}$, we can write $\mathbf{W}_{p_2}^l = \mathbf{W}_{p_2}^l \mathbf{U}_k \mathbf{U}_k^\top$. Substituting this in, we get

$$\nabla_l(\mathbf{X}_{\text{hidden}}^l) = \nabla_l(\mathbf{X}^{l+1}) \mathbf{U}_k \mathbf{U}_k^\top (\mathbf{W}_{p_2}^l \mathbf{U}_k \mathbf{U}_k^\top)^\top \quad (20)$$

$$= \nabla_l(\mathbf{X}^{l+1}) \mathbf{U}_k \mathbf{U}_k^\top \mathbf{U}_k \mathbf{U}_k^\top (\mathbf{W}_{p_2}^l)^\top. \quad (21)$$

Because \mathbf{U}_k is orthonormal, $\mathbf{U}_k^\top \mathbf{U}_k = \mathbf{I}$, so

$$\nabla_l(\mathbf{X}_{\text{hidden}}^l) = \nabla_l(\mathbf{X}^{l+1}) \mathbf{U}_k \mathbf{U}_k^\top (\mathbf{W}_{p_2}^l)^\top \quad (22)$$

$$= \nabla_l(\mathbf{X}^{l+1}) (\mathbf{W}_{p_2}^l \mathbf{U}_k \mathbf{U}_k^\top)^\top \quad (23)$$

$$= \nabla_l(\mathbf{X}^{l+1}) (\mathbf{W}_{p_2}^l)^\top. \quad (24)$$

This shows that projecting $\nabla_l(\mathbf{X}^{l+1})$ onto \mathcal{S} does not introduce any approximation error in $\nabla_l(\mathbf{X}_{\text{hidden}}^l)$.

Similarly, the remaining gradients are computed as follows. First,

$$\nabla_l(\mathbf{W}_1^l) = (\mathbf{X}_{\text{attn}}^l)^\top (\nabla_l(\mathbf{X}_{\text{hidden}}^l) \circ \nabla f_{\text{relu}}), \quad (25)$$

where $(\cdot \circ \cdot)$ denotes elementwise (Hadamard) multiplication. The gradient for $\mathbf{X}_{\text{attn}}^l$ then becomes

$$\nabla_l(\mathbf{X}_{\text{attn}}^l) = (\nabla_l(\mathbf{X}_{\text{hidden}}^l) \circ \nabla f_{\text{relu}}) (\mathbf{W}_1^l)^\top + (\nabla_l(\mathbf{X}_{\text{attn}}^l))^{\text{residual}} \quad (26)$$

$$= (\nabla_l(\mathbf{X}_{\text{hidden}}^l) \circ \nabla f_{\text{relu}}) (\mathbf{W}_1^l)^\top + \nabla_l(\mathbf{X}^{l+1}). \quad (27)$$

Next,

$$\nabla_l(\mathbf{W}_{p_1}^l) = (\mathbf{X}_{\text{concat}}^l)^\top \nabla_l(\mathbf{X}_{\text{attn}}^l) \quad (28)$$

$$= (\mathbf{X}_{\text{concat}}^l)^\top ((\nabla_l(\mathbf{X}_{\text{hidden}}^l) \circ \nabla f_{\text{relu}}) (\mathbf{W}_1^l)^\top + \nabla_l(\mathbf{X}^{l+1})). \quad (29)$$

Recall that $\text{Row}(\mathbf{W}_1^l) \subseteq \mathcal{S}$. Therefore, if $\nabla_l(\mathbf{X}^{l+1})$ is projected onto \mathcal{S} , it follows that $\text{Row}(\nabla_l(\mathbf{W}_{p_1}^l)) \subseteq \mathcal{S}$.

Finally, the update rule for $\mathbf{W}_{p_1}^l$ is

$$\mathbf{W}_{p_1}^l(t+1) = \mathbf{W}_{p_1}^l(t) - \gamma \nabla_l(\mathbf{W}_{p_1}^l(t)), \quad (30)$$

The gradient updates for $\mathbf{X}_{\text{concat}}^l$ involve

$$\mathbf{X}_{\text{concat}}^l = \mathbf{X}_{\text{attn}}^l (\mathbf{W}_{p_1}^l)^\top. \quad (31)$$

Because all gradients up to $\mathbf{X}_{\text{concat}}^l$ are preserved without loss, it follows via the chain rule that the gradient flow to the blocks below $\mathbf{X}_{\text{concat}}^l$ is also lossless. Consequently, when compressing $\nabla_L(\mathbf{X}^{l+1})$ as

$$(\nabla_L(\mathbf{X}^{l+1}))_{\text{compressed}} = \nabla_L(\mathbf{X}^{l+1}) \mathbf{U}_k \in \mathbb{R}^{b \times n \times k}, \quad (32)$$

we can fully recover it in the previous layer:

$$(\nabla_L(\mathbf{X}^{l+1}))_{\text{recovered}} = (\nabla_L(\mathbf{X}^{l+1}))_{\text{compressed}} \mathbf{U}_k^\top \quad (33)$$

$$= \nabla_L(\mathbf{X}^{l+1}), \quad (34)$$

thereby incurring no approximation error. Notably, since $k \ll d$, the compressed gradient $(\nabla_L(\mathbf{X}^{l+1}))_{\text{compressed}}$ is significantly lower-dimensional than $\nabla_L(\mathbf{X}^{l+1})$, yielding substantial communication savings over low-bandwidth links. This shows that by projecting gradients onto the same subspace used for forward-pass compression, we can achieve *lossless gradient compression* during backpropagation. This strategy further eliminates the need for frequent projections of $\mathbf{W}_{p_2}^l$ maintaining its confinement to the intended subspace.

D On the error accumulation of lossy compressions

A key difference between Distributed Data Parallel (DDP) training and Model Parallel (MP) training lies in how gradients are exchanged and how compression is applied. In DDP, gradients of model parameters are exchanged after each training step, enabling compression to be applied across the entire gradient vector in one operation. In contrast, PP training requires the exchange of activations and activation gradients between model partitions, leading to a layer-wise compression approach. Unlike DDP, where compression affects the gradient as a whole, MP training introduces independent compression errors at each layer. These errors accumulate progressively during training, impacting gradient computations layer by layer and posing unique challenges in maintaining model accuracy and stability.

Given these distinctions, it is critical to analyze MP training within the constraints of activation gradient compression. The layer-by-layer compression in MP can lead to compounded errors that significantly influence convergence behavior and overall model performance. The following result focuses on investigating the relative error with respect to the weight gradients, occurred by a compression error induced by a particular layer, on an arbitrary layer below it.

Theorem D.1. *Consider a feedforward neural network with L layers, where layer l applies a (differentiable) function*

$$x_{l+1} = f_l(x_l), \quad l = 1, \dots, L.$$

Let $\nabla_L(x_l)$ denote the gradient of the final loss \mathcal{L} with respect to the layer's input x_l . Suppose that:

1. *The spectral norm of the Jacobian $\nabla f_l(x_l)$ is bounded above by $\nu > 0$ for all l , i.e. $\|\nabla f_l(x_l)\| \leq \nu$.*
2. *In backpropagation, an additional error e_l is introduced at each layer l , with $\|e_l\| \leq e$ for some constant $e > 0$.*

Define ε_l to be the cumulative error in the gradient at layer l . Then for $\nu > 1$, ε_l can grow exponentially with the total number of layers L ; in particular,

$$\|\varepsilon_l\| \leq e \frac{\nu^{L-l+1} - 1}{\nu - 1},$$

which is an exponential function of L when $\nu > 1$.

Proof. Recall the usual chain rule for the gradient of the final loss \mathcal{L} with respect to the input x_l of layer l :

$$\nabla_L(x_l) = \nabla_L(x_{l+1}) \nabla f_l(x_l).$$

Assume that at each layer we introduce an error in the gradient. Let

$$\varepsilon_l = (\text{true gradient at layer } l) - (\text{observed/propagated gradient at layer } l).$$

When moving from layer l to layer $l - 1$, the error recursion becomes:

$$\varepsilon_{l-1} = \varepsilon_l \nabla f_{l-1}(x_{l-1}) + e_{l-1},$$

where e_{l-1} is the *newly introduced* error at layer $l - 1$. Unfolding this backwards gives a general expansion:

$$\varepsilon_l = e_l + \sum_{j=l+1}^L \left(\prod_{i=l}^{j-1} \nabla f_i(x_i) \right) e_j.$$

Taking the norm and using the assumption $\|\nabla f_i(x_i)\| \leq \nu$ and $\|e_j\| \leq e$, we get:

$$\|\varepsilon_l\| \leq \sum_{j=l}^L \left(\prod_{i=l}^{j-1} \|\nabla f_i(x_i)\| \right) \|e_j\| \leq \sum_{j=l}^L \nu^{j-l} e = e \sum_{k=0}^{L-l} \nu^k,$$

where $k = j - l$. This geometric sum is

$$\sum_{k=0}^{L-l} \nu^k = \frac{\nu^{L-l+1} - 1}{\nu - 1} \quad (\text{valid for } \nu \neq 1).$$

Hence,

$$\|\varepsilon_l\| \leq e \frac{\nu^{L-l+1} - 1}{\nu - 1}.$$

Since $\nu > 1$, ν^{L-l+1} grows exponentially in L . □

This is an important result. In particular, this result indicates that in cases where the spectral norm of the weight matrices is large enough, the upper bound for the error for the lower layers can grow exponentially as the depth of the network increases.

E Vanishing of gradient updates outside a subspace

The result in this section highlights an important property of AdamW: the optimizer's weight updates progressively align with the subspace in which the gradients are constrained. In particular, if the gradients of a weight matrix lie within a specific low-dimensional subspace \mathcal{S} , the updates made to the weight matrix are increasingly confined to \mathcal{S} over time. This behavior ensures that any weight components orthogonal to \mathcal{S} are asymptotically suppressed, with the weight matrix ultimately converging to \mathcal{S} . To this end, we first prove Lemma E.1 and then Theorem E.2.

Lemma E.1 (Orthogonal Distortion Bound). *Let $a = (a_1, \dots, a_n) \in \mathbb{R}^n$ be a nonzero vector, and let $v_1, \dots, v_n > 0$ be positive scalars. Define*

$$m = \min_{1 \leq i \leq n} v_i, \quad M = \max_{1 \leq i \leq n} v_i, \quad \Gamma = \frac{M}{m}.$$

Form the vector

$$b = (v_1 a_1, v_2 a_2, \dots, v_n a_n) \in \mathbb{R}^n,$$

and let b_\perp be the component of b orthogonal to a . Then the following two bounds hold:

$$\|b_\perp\| \leq \frac{M - m}{2} \|a\| = \frac{m(\Gamma - 1)}{2} \|a\| \quad \text{and} \quad \|b_\perp\| \leq \frac{\Gamma - 1}{2} \|b\|.$$

Proof. First, decompose b into its projection onto a plus its component orthogonal to a :

$$b = \text{proj}_a(b) + b_\perp, \quad \text{where } b_\perp \perp a.$$

The norm of b_\perp can be expressed by the well-known identity

$$\|b_\perp\|^2 = \|b\|^2 - \frac{(b \cdot a)^2}{\|a\|^2}.$$

Since $b = (v_1 a_1, \dots, v_n a_n)$, we have

$$\|b\|^2 = \sum_{i=1}^n (v_i a_i)^2 = \sum_{i=1}^n v_i^2 a_i^2, \quad \text{and} \quad b \cdot a = \sum_{i=1}^n v_i a_i^2.$$

Thus

$$\|b_{\perp}\|^2 = \sum_{i=1}^n v_i^2 a_i^2 - \frac{\left(\sum_{i=1}^n v_i a_i^2\right)^2}{\sum_{i=1}^n a_i^2}.$$

Let $A^2 = \|a\|^2 = \sum_{i=1}^n a_i^2$. Define weights

$$s_i = \frac{a_i^2}{A^2} \quad \text{so that} \quad s_1 + s_2 + \cdots + s_n = 1.$$

Then

$$\sum_{i=1}^n v_i^2 a_i^2 = A^2 \sum_{i=1}^n v_i^2 s_i, \quad \sum_{i=1}^n v_i a_i^2 = A^2 \sum_{i=1}^n v_i s_i.$$

Hence

$$\|b_{\perp}\|^2 = A^2 \left[\sum_{i=1}^n v_i^2 s_i - \left(\sum_{i=1}^n v_i s_i \right)^2 \right].$$

Recognize that

$$\sum_{i=1}^n v_i^2 s_i - \left(\sum_{i=1}^n v_i s_i \right)^2$$

is precisely the variance $\text{Var}_s(V)$ of the values v_i with respect to the probability distribution $\{s_i\}$. Since $m \leq v_i \leq M$ for all i , the variance of any random variable confined to $[m, M]$ is at most $\frac{(M-m)^2}{4}$. (This is realized, for instance, by taking a two-point distribution at m and M with equal probabilities.) Therefore,

$$\sum_{i=1}^n v_i^2 s_i - \left(\sum_{i=1}^n v_i s_i \right)^2 \leq \frac{(M-m)^2}{4}.$$

Consequently,

$$\|b_{\perp}\|^2 \leq A^2 \cdot \frac{(M-m)^2}{4} = \frac{(M-m)^2}{4} \|a\|^2,$$

and hence

$$\|b_{\perp}\| \leq \frac{M-m}{2} \|a\|.$$

Substituting $M = m\Gamma$ (so $M-m = m(\Gamma-1)$) gives

$$\|b_{\perp}\| \leq \frac{m(\Gamma-1)}{2} \|a\|.$$

For the alternative form in terms of $\|b\|$, note that

$$\|b\| = \sqrt{\sum_{i=1}^n (v_i a_i)^2} \geq \sqrt{\sum_{i=1}^n m^2 a_i^2} = m \|a\|.$$

Hence $\|a\| \leq \frac{1}{m} \|b\|$, and

$$\|b_{\perp}\| \leq \frac{(M-m)}{2} \|a\| \leq \frac{(M-m)}{2m} \|b\| = \frac{\Gamma-1}{2} \|b\|.$$

This completes the proof. \square

Theorem E.2. Let $\mathbf{W}(t) \in \mathbb{R}^{m \times n}$ be the weight matrix of a neural network optimized using AdamW at the t -th iteration. Suppose the rows of the gradient matrix $\nabla_L(\mathbf{W}(t))$ lie within a subspace $\mathcal{S} \subseteq \mathbb{R}^n$. Let Δ_t be defined as

$$\mathbf{W}(t+1) = \mathbf{W}(t) - \eta(\Delta_t + \alpha \mathbf{W}(t)) \tag{35}$$

where η is the learning rate and,

$$\Delta_t = \mathbf{M}_t \oslash (\sqrt{\mathbf{V}} + \epsilon) \quad (36)$$

where $(\cdot \oslash \cdot)$ denotes the element wise Hadamard division and $(\alpha, \epsilon) > 0$ where α is the weight decay and ϵ is a constant added to avoid division by 0. \mathbf{M}_t is the first momentum matrix defined as

$$\mathbf{M}_t = \beta_1 \mathbf{M}_{t-1} + (1 - \beta_1) \nabla_L(\mathbf{W}(t)) \quad (37)$$

and \mathbf{V} is the second moment,

$$\mathbf{V}_t = \beta_2 \mathbf{V}_{t-1} + (1 - \beta_2) \nabla_L(\mathbf{W}(t))^2 \quad (38)$$

Also, let Δ_t^\perp be the matrix whose rows consist of the components orthogonal to \mathcal{S} from the rows of Δ_t . Assume that g_t^2 follows a sub-Gaussian distribution with a variance proxy $(\sigma(t))^2$ where g_t is any element of $\nabla_L(\mathbf{W}(t))$. Then, with probability of at least $1 - \delta$, $\lim_{t \rightarrow \infty} \frac{\|\Delta_t^\perp\|}{\|\Delta_t\|} = 0$ if $\lim_{t \rightarrow \infty} \sigma(t) = 0$.

Proof. Since the first-moment estimate \mathbf{M}_t in AdamW remains a (biased) linear combination of past gradients $\nabla_L(\mathbf{W}(t))$, its rows stays within \mathcal{S} (recall that vector spaces are closed under addition). However, the rows of second-moment estimate \mathbf{V}_t are not constant-value vectors, and thus, the element-wise division of \mathbf{M}_t by $(\sqrt{\mathbf{V}} + \epsilon)$ distorts and pushes the rows of \mathbf{M}_t outside \mathcal{S} . Consequently, the rows of Δ_t are not confined to \mathcal{S} . Our goal is to show that $\frac{\|\Delta_t^\perp\|}{\|\Delta_t\|}$ asymptotically goes to zero with the variance decay.

Step 1: Bounding the second-moment ratio. Since the distribution of g_t^2 is sub-Gaussian, then for each g_t , we have

$$P(g_t^2 \geq x) \leq 2 \exp\left(-\frac{x^2}{2\sigma^2}\right). \quad (39)$$

Note that we drop the coordinate indexes (i, j) from g_t for brevity. Using a union bound over $m \times n$ coordinates and at any given time t , we have

$$P(\max_{i,j} g_t^2 \geq x) \leq 2mn \exp\left(-\frac{x^2}{2(\sigma(t))^2}\right) \quad (40)$$

Then, with probability at least $1 - \delta$, we get

$$\max_{i,j} (g_t^2) \leq \sigma(t) \sqrt{2 \ln\left(\frac{2mn}{\delta}\right)} \quad (41)$$

Since any element v_t of \mathbf{V}_t is a moving average of g_t^2 , we can expand it as,

$$v_t = \beta_2^{(t-1)}(1 - \beta_2)g_1^2 + \beta_2^{(t-2)}(1 - \beta_2)g_2^2 + \dots + \beta_2^2(1 - \beta_2)g_{t-2}^2 + \beta_2(1 - \beta_2)g_{t-1}^2 + (1 - \beta_2)g_t^2 \quad (42)$$

(assuming $v_0 = 0$). Note that for large t , we can have a $\tau(k)$ such that $v_t - \tau(k) < \mu$ for arbitrary small μ , where

$$\tau(k) = \beta_2^{(t-k)}(1 - \beta_2)g_k^2 + \beta_2^{(t-k-1)}(1 - \beta_2)g_{k+1}^2 + \dots + \beta_2^2(1 - \beta_2)g_{t-2}^2 + \beta_2(1 - \beta_2)g_{t-1}^2 + (1 - \beta_2)g_t^2 \quad (43)$$

Plugging the above result to Eq. 41, with probability at least $1 - \delta$, we again have

$$\max_{i,j} [v_t]_{i,j} \leq \max_{t \in [t-k, T]} \sigma(t) \sqrt{2 \ln \left(\frac{2mn}{\delta} \right)} \quad (44)$$

Then,

$$\frac{\sqrt{\max_{i,j} [v_t]_{i,j}} + \epsilon}{\sqrt{\min_{i,j} [v_t]_{i,j}} + \epsilon} \leq 1 + \frac{\sqrt{\max_{i,j} [v_t]_{i,j}}}{\epsilon} \leq 1 + \frac{\sqrt{\max_{t \in [t-k, T]} \sigma(t) \sqrt{2 \ln \left(\frac{2mn}{\delta} \right)}}}{\epsilon}, \quad (45)$$

since $\sqrt{\min_{i,j} [v_t]_{i,j}} = 0$. Let $\kappa(t) = \frac{\sqrt{\max_{t \in [t-k, T]} \sigma(t) \sqrt{2 \ln \left(\frac{2mn}{\delta} \right)}}}{\epsilon}$. Then, the coordinate-wise learning-rate scaling $\frac{1}{\sqrt{v_t + \epsilon}}$ cannot differ among coordinates by more than $1 + \kappa(t)$ with a high probability.

Step 2: Relating off-subspace updates to the ratio. Consider any row vector of Δ_t to be $(\Delta_t)_{i,:}$. Then, by Applying the update Eq. 45 to Lemma 1, and by definition of Δ_t , we have

$$\|(\Delta_t)_{i,:}^\perp\| \leq \frac{\kappa(t)}{2} \|(\Delta_t)_{i,:}\| \quad (46)$$

Step 3: Taking the limit $t \rightarrow \infty$. Because we have

$$\lim_{t \rightarrow \infty} \sigma(t) = 0, \quad (47)$$

trivially, we also have

$$\lim_{T \rightarrow \infty} \max_{t \in [T-k, T]} \sigma(t) = 0. \quad (48)$$

Then, it follows from the definition of $\kappa(t)$ that

$$\lim_{t \rightarrow \infty} \kappa(t) = 0. \quad (49)$$

Therefore, from Eq. 46, assuming $\|(\Delta_t)_{i,:}\| \neq 0$ we finally have

$$0 \leq \lim_{t \rightarrow \infty} \frac{\|(\Delta_t)_{i,:}^\perp\|}{\|(\Delta_t)_{i,:}\|} \leq 0 \quad (50)$$

$$= \lim_{t \rightarrow \infty} \frac{\|(\Delta_t)_{i,:}^\perp\|}{\|(\Delta_t)_{i,:}\|} = 0 \quad (51)$$

In simpler terms, the distortion of the rows of Δ_t outside \mathcal{S} is ultimately goes to zero.

□

Discussion: Above theorem gives us an important insight. In particular, it ensures that if the gradients of an unrestricted network predominantly lie in a specific low-dimensional subspace \mathcal{S} , then the AdamW optimizer restricts updates outside that subspace. On the other hand, as training progresses, gradient norms decrease and become more uniform across coordinates, which has been rigorously proved in previous works within reasonable bounds [25, 59, 9]. Consequently, the variance of the

gradients also tends to diminish, justifying our assumption of $\lim_{t \rightarrow \infty} \sigma(t) \rightarrow 0$. This reduction further curtails the extent of any *off-subspace* updates, thereby indicating that AdamW naturally remains focused on the directions most important to learning.

Further, empirical evidence supports the notion that gradients of unconstrained networks often lie predominantly within a low-dimensional subspace (we also empirically validate this in Fig. 7). This observation aligns with the hypothesis that many deep learning problems exhibit significant redundancy in parameter updates, with key directions of optimization being confined to a smaller subspace. By leveraging this inherent structure, it becomes possible to guide optimization dynamics in a manner that preserves computational efficiency while maintaining convergence. We conduct an extended discussion next.

E.1 On the assumptions of Theorem E.2

For Theorem E.2, we employ two assumptions 1) the gradients of weight matrices are confined to a low-dimensional subspace and 2) the variance of the gradients diminish as the model converges. Next, we provide justifications for these assumptions.

E.1.1 The gradients of weight matrices are confined to a low-dimensional subspace

To empirically validate this behavior, we train an 8-layer network on C4 and track the stable rank progression of its projection matrices throughout training. As shown in Fig. 7, the gradients maintain an exceptionally low stable rank relative to the maximum possible rank from initialization to convergence. This confirms that the gradients primarily focus on a few dominant directions.

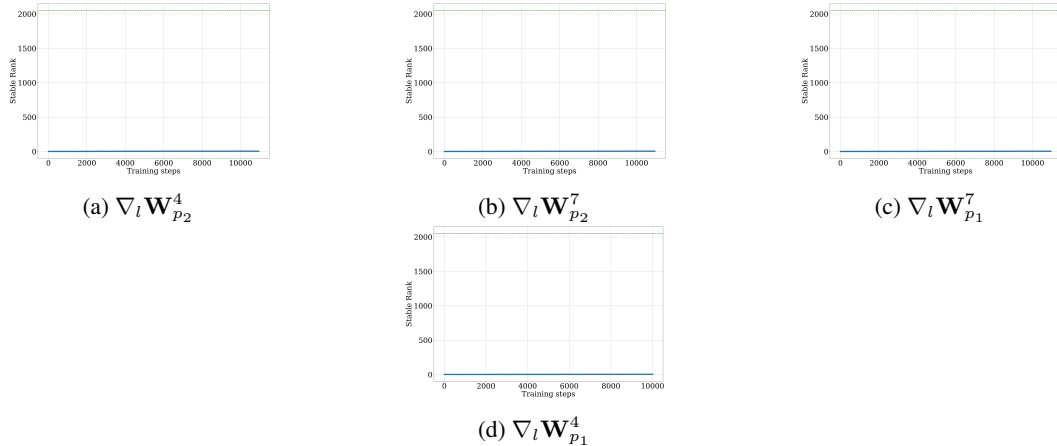


Figure 7: Low dimensionality of the gradients of projection matrices. We measure the stable ranks of gradients of the projection matrices over the course of training. In each plot, the green dashed line indicates the maximum rank. As shown, all the weight matrices consistently maintain a significantly lower rank ($\sim < 5$) throughout the training. This validates the assumption of our theorem E.2 that the weight gradients of the projection matrices are confined to a low-dimensional subspace.

The assumption that weight gradients span a low-dimensional subspace is well-supported by prior research. [16] demonstrated that gradient descent in deep learning does not explore the full parameter space but instead operates within a constrained subspace, largely dictated by the spectral structure of the Hessian. The Hessian matrix features a dominant subspace associated with large eigenvalues, where most gradient updates occur, and a bulk subspace characterized by near-zero eigenvalues that correspond to flat directions in the loss landscape. Empirical evidence indicates that early in training, gradients align with this dominant Hessian subspace and remain confined within it throughout the optimization process.

This phenomenon extends to the learned representations themselves. [60] showed that gradient descent naturally gravitates toward low-rank solutions, a behavior arising from implicit regularization within the optimization process. Instead of initializing with full-rank representations, training typically begins in a low-rank regime and expands rank as necessary. This trend has been consistently observed

across multiple architectures, including MLPs, CNNs, and transformers. The concept of spectral bias further supports this observation, suggesting that neural networks initially learn broad, low-rank structures before refining finer details.

A practical application of this low-rank property is evident in [7]’s introduction of Low-Rank Gradient Descent (LRGD), which explicitly leverages low-rank subspaces to expedite training. By identifying dominant gradient directions and confining updates to this reduced subspace, LRGD significantly lowers computational costs without sacrificing optimization effectiveness. Theoretical results demonstrate that in strongly convex settings, LRGD reduces oracle complexity from $O(p \log(1/\epsilon))$ to $O(r \log(1/\epsilon) + rp)$, where $r \ll p$. Similar improvements are observed in non-convex environments. Moreover, [54] illustrated that in deep networks trained with cross-entropy loss, the Hessian spectrum exhibits a clear separation. The top subspace, whose dimensionality approximates the number of classes in the dataset, captures the majority of gradient activity. In contrast, the bulk subspace, filled with small eigenvalues, corresponds to nearly flat directions in the loss landscape. This structure implies that significant changes in the loss function occur primarily within the top subspace. Empirical findings also reveal that gradients quickly align with the top Hessian subspace early in training and persist within it. This effect has been documented across diverse architectures such as fully connected networks, convolutional networks, ResNets, and transformers. While individual Hessian eigenvectors evolve during training, the dominant subspace remains relatively stable, guiding the optimization path. The low-rank nature of gradients is further validated by [47], who observed a rapid decay in the singular values of gradient matrices. This decay implies that the most meaningful updates are confined to a small, structured subspace. Additionally, the gradient covariance matrix in deep networks shows exponential spectral decay, suggesting that full-rank updates are not essential.

Above findings provide robust empirical and theoretical support for our low-rank gradient assumption, directly motivating our approach. By exploiting low-rank constraints, we achieve substantial reductions in both computation and communication overhead while preserving performance. Given that deep learning models inherently exhibit implicit regularization, explicitly enforcing a low-rank structure in our compression could even induce a beneficial regularization effect—potentially enhancing performance beyond that of non-compressed models, as indicated in Table 1.

E.1.2 The variance of the gradients diminish as the model converges.

The reduction in gradient variance as a model converges is a well-observed phenomenon in optimization literature. As training progresses, stochastic gradient estimates become more stable, leading to smoother updates and accelerated convergence. This effect is particularly pronounced in adaptive learning rate optimizers such as Adam [20] and AdamW [32]. In particular, [3] showed that these methods inherently reduce the variance of parameter updates over time, contributing to stable convergence. [31] further refined this idea by introducing a variant of Adam designed to enforce adaptive variance reduction, ensuring that the variance of the stochastic gradient estimator decreases as the model approaches optimality.

Beyond adaptive optimization, variance reduction techniques have been widely studied in the context of stochastic gradient descent (SGD). Traditional SGD suffers from high variance in gradient estimates, particularly in early training stages, which can hinder convergence. To mitigate this, methods such as Stochastic Variance Reduced Gradient (SVRG) and SAGA [49] have been proposed. These approaches reduce variance by incorporating control variates or maintaining a memory of past gradients, yielding more accurate updates and improved convergence rates. Additional techniques, including Kalman-based filtering [48] and stochastic filtering methods [57], further enhance stability by adaptively reducing noise in gradient estimates.

Normalization layers have also been shown to smooth parameter spaces and gradients, effectively reducing variance. This smoothness, characterized by a lower Lipschitz constant, results in more predictive gradients, which can be interpreted as a form of implicit variance reduction. The effect is particularly beneficial in deep networks, where sharp loss landscapes can otherwise lead to erratic updates.

In summary, diminishing gradient variance is a natural outcome of optimization dynamics. As training progresses and model parameters approach optimality, stochastic gradients become increasingly stable, even in the absence of explicit variance reduction techniques. This effect can be attributed to the curvature of the loss landscape: near an optimal solution, the loss function tends to be flatter, leading to more consistent gradient estimates across mini-batches. Consequently, as the model

converges, gradient variance inherently diminishes, reinforcing the validity of this assumption in our work.

F Validation at compute-optimal

We conduct additional training experiments to demonstrate the effectiveness of our proposed compression method at the compute-optimal point of model training. The compute-optimal point, as defined by prior work such as Chinchilla, represents an ideal trade-off between model size and token count, optimizing performance within a fixed computational budget. Typically, for architectures like LLaMA, this optimal point corresponds to approximately a 1:20 model-to-token ratio. Consequently, we trained a 640M-parameter model for around 12 billion tokens, aligning closely with the compute-optimal recommendation.

The training was executed using 8 FSDP workers, achieving approximately 22k tokens per second throughput. The results, summarized in Table 4, demonstrate that our compression method not only matches but marginally outperforms the non-compressed baseline in terms of validation perplexity.

We further trained a 2.3B parameter model for 130 billion tokens (well past the Chinchilla optimal) and measured the validation loss. The results are shown in Table 3.

Table 3: Validation perplexity (lower is better) at (130 billion tokens) for a 2.3B-parameter model. Our compression has minimal reduction in performance compared to the uncompressed model.

Model	Fineweb-Edu
Baseline (non-compressed)	10.59
Ours (compressed)	12.06

Table 4: Validation perplexity (lower is better) at the compute-optimal point (12 billion tokens) for a 640M-parameter model.

Model	C4	BookCorpus
Baseline (non-compressed)	12.61	12.79
Ours (compressed)	12.53	12.67

These results confirm that our proposed compression method maintains or slightly improves model performance at the crucial compute-optimal training regime.

G Continuous-Time Gradient Flow with Weight Decay Converges to a Low-Dimensional Subspace

The result in this section reveals a fundamental property of optimization under AdamW: the optimizer inherently drives weight matrices to converge asymptotically to a shared subspace, regardless of their initial values. This behavior stems from the decoupled weight decay mechanism, which systematically suppresses components of the weight matrix that receive negligible or noisy gradient updates during training. Over time, the weight matrix becomes confined to a low-dimensional subspace spanned by the significant directions of the gradient updates, leading to an effectively low-rank representation.

Theorem G.1. Consider a continuous-time dynamics for $\mathbf{W}(t) \in \mathbb{R}^{m \times n}$ governed by

$$\frac{d\mathbf{W}}{dt} = -\eta \left(\Delta_t + \alpha \mathbf{W}(t) \right),$$

where $\eta, \alpha > 0$ are constants, and Δ_t represents the gradient-like update without weight decay (i.e. the “AdamW” part, ignoring the $\alpha \mathbf{W}(t)$ term). Assume that for large t , Δ_t has its rows confined predominantly to a subspace $\mathcal{S} \subseteq \mathbb{R}^n$ (so that the orthogonal component of Δ_t vanishes asymptotically). Then, as $t \rightarrow \infty$, the component of $\mathbf{W}(t)$ lying in \mathcal{S}^\perp decays exponentially to zero. Consequently, the rows of the final solution $\mathbf{W}(\infty)$ lie in (or arbitrarily close to) \mathcal{S} .

Proof. Let $\Pi_{\mathcal{S}}$ denote the orthogonal projector onto \mathcal{S} (applied rowwise to any $m \times n$ matrix). Decompose:

$$\mathbf{W}(t) = \mathbf{W}_{\parallel}(t) + \mathbf{W}_{\perp}(t), \quad \text{where} \quad \mathbf{W}_{\parallel}(t) = \Pi_{\mathcal{S}}[\mathbf{W}(t)], \quad \mathbf{W}_{\perp}(t) = (\mathbf{I} - \Pi_{\mathcal{S}})\mathbf{W}(t).$$

By hypothesis (see also Theorem E.2), the rows of Δ_t become confined to \mathcal{S} as $t \rightarrow \infty$, i.e.

$$\lim_{t \rightarrow \infty} (\mathbf{I} - \Pi_{\mathcal{S}})\Delta_t = 0.$$

Thus, for large t , the orthogonal component of Δ_t is negligible. Project the ODE onto \mathcal{S}^{\perp} :

$$\frac{d\mathbf{W}_{\perp}(t)}{dt} = (\mathbf{I} - \Pi_{\mathcal{S}})\frac{d\mathbf{W}}{dt} = -\eta(\mathbf{I} - \Pi_{\mathcal{S}})(\Delta_t + \alpha \mathbf{W}(t)).$$

For sufficiently large t , $(\mathbf{I} - \Pi_{\mathcal{S}})\Delta_t$ is negligible, so asymptotically,

$$\frac{d\mathbf{W}_{\perp}(t)}{dt} \approx -\eta\alpha(\mathbf{I} - \Pi_{\mathcal{S}})(\mathbf{W}(t)) = -\eta\alpha\mathbf{W}_{\perp}(t).$$

This is a linear ODE:

$$\frac{d\mathbf{W}_{\perp}(t)}{dt} = -\eta\alpha\mathbf{W}_{\perp}(t),$$

whose solution is

$$\mathbf{W}_{\perp}(t) = \exp(-\eta\alpha t)\mathbf{W}_{\perp}(0).$$

Hence $\mathbf{W}_{\perp}(t)$ decays exponentially to $\mathbf{0}$. Therefore,

$$\mathbf{W}(t) = \mathbf{W}_{\parallel}(t) + \mathbf{W}_{\perp}(t) \rightarrow \mathbf{W}_{\parallel}(t), \quad \text{with} \quad \mathbf{W}_{\parallel}(t) \in \mathcal{S}.$$

Thus, as $t \rightarrow \infty$, the rows of $\mathbf{W}(t)$ lie in \mathcal{S} (or arbitrarily close), proving the claim. \square

Discussion: Unlike standard stochastic gradient descent (SGD), which applies weight decay as a modification to the gradient updates, AdamW separates the weight decay term from the gradient computation. This decoupled weight decay has two key effects: 1.) **Suppression of Insignificant Components:** Components of the weight matrix orthogonal to the subspace spanned by significant gradient updates decay exponentially due to weight decay. This ensures that the optimization focuses on meaningful directions of the gradient. 2) **Low-Dimensional Convergence:** By amplifying the impact of directions where the gradient signal is strongest, the optimizer effectively projects the weight matrix onto a subspace defined by these directions, leaving other components asymptotically negligible.

To empirically validate this property, we analyzed the spectral structure of weight matrices trained under AdamW. As shown in Fig. 1, the spectrum of these matrices exhibits rapid decay, with the majority of the spectral mass concentrated in a few dominant directions.

H Convergence of Partially-Projected Gradient Descent

This section establishes that constraining a subset of weights in a neural network to lie within a low-dimensional subspace does not degrade the theoretical convergence guarantees of constrained optimization. Specifically, the network still achieves convergence to a first-order stationary point at the rate of $O(1/T)$, where T is the number of optimization steps. This is consistent with standard results in constrained optimization and provides a preliminary theoretical foundation for employing subspace constraints in practical applications.

Proposition H.1. *Let $f(\mathbf{W}^{(s)}, \mathbf{W}^{(u)})$ be a possibly non-convex function that is L -smooth in all its parameters $\mathbf{W} = (\mathbf{W}^{(s)}, \mathbf{W}^{(u)})$, where $\mathbf{W}^{(s)} \in \mathbb{R}^k$ is a constrained block and $\mathbf{W}^{(u)} \in \mathbb{R}^d$ is unconstrained and $k < d$. Suppose we have a closed subspace $\mathcal{S} \subseteq \mathbb{R}^k$, and we define the feasible set*

$$\mathbf{W} = \mathcal{S} \times \mathbb{R}^{d_u}.$$

Consider the following partially-projected gradient step:

$$\widetilde{\mathbf{W}}_{t+1}^{(s)} = \mathbf{W}_t^{(s)} - \eta \nabla_{(s)} f(\mathbf{W}_t), \quad \widetilde{\mathbf{W}}_{t+1}^{(u)} = \mathbf{W}_t^{(u)} - \eta \nabla_{(u)} f(\mathbf{W}_t),$$

$$\mathbf{W}_{t+1}^{(s)} = \Pi_{\mathcal{S}}(\widetilde{\mathbf{W}}_{t+1}^{(s)}), \quad \mathbf{W}_{t+1}^{(u)} = \widetilde{\mathbf{W}}_{t+1}^{(u)},$$

where $\Pi_{\mathcal{S}}(\cdot)$ is the Euclidean projection onto \mathcal{S} , $\nabla_{(s)}f$ and $\nabla_{(u)}f$ are the gradients of f w.r.t. the constrained and unconstrained blocks, respectively, and $\eta > 0$ is a fixed step size.

Then, for any $T \geq 1$ and $\eta \leq \frac{1}{L}$, the iterates $\{\mathbf{W}_t\}$ satisfy the following stationary guarantee:

$$\min_{0 \leq t < T} \|\nabla f(\mathbf{W}_t)\|^2 \leq \frac{2(f(\mathbf{W}_0) - f^*)}{\eta T}$$

where $f^* = \min_{\mathbf{W} \in \mathcal{W}} f(\mathbf{W})$ and \mathbf{W}_0 is the initial parameter vector.

Proof. Since $f(\mathbf{W}^{(s)}, \mathbf{W}^{(u)})$ is L -smooth in all parameters, for any \mathbf{W}, \mathbf{W}' we have

$$f(\mathbf{W}') \leq f(\mathbf{W}) + \nabla f(\mathbf{W})^\top (\mathbf{W}' - \mathbf{W}) + \frac{L}{2} \|\mathbf{W}' - \mathbf{W}\|^2. \quad (52)$$

Setting $\mathbf{W} = \mathbf{W}_t$ and $\mathbf{W}' = \mathbf{W}_{t+1}$, we get

$$f(\mathbf{W}_{t+1}) \leq f(\mathbf{W}_t) + \nabla f(\mathbf{W}_t)^\top (\mathbf{W}_{t+1} - \mathbf{W}_t) + \frac{L}{2} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2. \quad (53)$$

Recall the update:

$$\mathbf{W}_{t+1}^{(s)} = \Pi_{\mathcal{S}}(\mathbf{W}_t^{(s)} - \eta \nabla_{(s)}f(\mathbf{W}_t)), \quad \mathbf{W}_{t+1}^{(u)} = \mathbf{W}_t^{(u)} - \eta \nabla_{(u)}f(\mathbf{W}_t). \quad (54)$$

$$\mathbf{W}_{t+1} = \left(\Pi_{\mathcal{S}}(\mathbf{W}_t^{(s)} - \eta \nabla_{(s)}f(\mathbf{W}_t)), \mathbf{W}_t^{(u)} - \eta \nabla_{(u)}f(\mathbf{W}_t) \right). \quad (55)$$

Projecting the $\widetilde{\mathbf{W}}_{t+1}^{(s)}$ block onto \mathcal{S} , is a 1-Lipschitz (nonexpansive) operation in the full Euclidean space. Concretely, consider $\|(\Pi_{\mathcal{S}}(\widetilde{\mathbf{W}}_{t+1}^{(s)}) - \mathbf{W}_t^{(s)})\|$. Also recall that $\mathbf{W}_t^{(s)} \in \mathcal{S}$. It follows from the classical projection lemma that,

$$\|(\mathbf{W}_{t+1}^{(s)} - \mathbf{W}_t^{(s)})\| = \|(\Pi_{\mathcal{S}}(\widetilde{\mathbf{W}}_{t+1}^{(s)}) - \mathbf{W}_t^{(s)})\| \leq \|(\widetilde{\mathbf{W}}_{t+1}^{(s)} - \mathbf{W}_t^{(s)})\| \quad (56)$$

This happens because the projection minimizes the Frobenius norm to the subset, so the projected point is closer than the unprotected one.

We also leave the disjoint set $\mathbf{W}^{(u)}$ untouched. So we have,

$$\widetilde{\mathbf{W}}_{t+1}^{(u)} = \mathbf{W}_{t+1}^{(u)} \quad (57)$$

Since $\mathbf{W}^{(s)}, \mathbf{W}^{(u)}$ are disjoint blocks, We can combine the blocks (Pythagorean identity): The Frobenius norm is the Euclidean norm on the concatenated vector, so

$$\|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 = \|(\mathbf{W}_{t+1}^{(s)} - \mathbf{W}_t^{(s)})\|^2 + \|(\mathbf{W}_{t+1}^{(u)} - \mathbf{W}_t^{(u)})\|^2$$

From Eq. 56 and 57 we have,

$$\|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \leq \|(\widetilde{\mathbf{W}}_{t+1}^{(s)} - \mathbf{W}_t^{(s)})\|^2 + \|(\widetilde{\mathbf{W}}_{t+1}^{(u)} - \mathbf{W}_t^{(u)})\|^2 \quad (58)$$

As $\mathbf{W}^{(s)}, \mathbf{W}^{(u)}$ are disjoint, we get,

$$\|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \leq \|\widetilde{\mathbf{W}}_{t+1} - \mathbf{W}_t\|^2 \quad (59)$$

$$\|\mathbf{W}_{t+1} - \mathbf{W}_t\| \leq \|\widetilde{\mathbf{W}}_{t+1} - \mathbf{W}_t\| \quad (60)$$

$(\widetilde{\mathbf{W}}_{t+1} - \mathbf{W}_t)$ is the update step which is equal to $\eta(\nabla f(\mathbf{W}_t))$. Thus,

$$\|\mathbf{W}_{t+1} - \mathbf{W}_t\| \leq \eta \|\nabla f(\mathbf{W}_t)\|. \quad (61)$$

Plugging this bound back into the smoothness inequality (Eq. 53), we can get

$$f(\mathbf{W}_{t+1}) \leq f(\mathbf{W}_t) + \nabla f(\mathbf{W}_t)^\top (\mathbf{W}_{t+1} - \mathbf{W}_t) + \frac{L}{2} \eta^2 \|\nabla f(\mathbf{W}_t)\|^2. \quad (62)$$

Further, from the optimality condition, it follows that,

$$(\mathbf{W}_{t+1} - (\mathbf{W}_t - \eta \nabla f(\mathbf{W}_t)))^\top (\mathbf{W}_{t+1} - \mathbf{W}_t) \leq 0 \quad (63)$$

$$(\eta \nabla f(\mathbf{W}_t))^\top (\mathbf{W}_{t+1} - \mathbf{W}_t) \leq -\|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \quad (64)$$

$$\nabla f(\mathbf{W}_t)^\top (\mathbf{W}_{t+1} - \mathbf{W}_t) \leq -\frac{1}{\eta} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \quad (65)$$

Substituting above in Eq. 62,

$$f(\mathbf{W}_{t+1}) \leq f(\mathbf{W}_t) - \frac{1}{\eta} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 + \frac{L}{2} \eta^2 \|\nabla f(\mathbf{W}_t)\|^2. \quad (66)$$

$$f(\mathbf{W}_{t+1}) \leq f(\mathbf{W}_t) - \frac{1}{\eta} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 + \frac{L}{2} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2. \quad (67)$$

$$f(\mathbf{W}_{t+1}) \leq f(\mathbf{W}_t) + \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \left(\frac{L}{2} - \frac{1}{\eta} \right). \quad (68)$$

$$f(\mathbf{W}_{t+1}) - f(\mathbf{W}_t) \leq \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2 \left(\frac{L}{2} - \frac{1}{\eta} \right). \quad (69)$$

$$f(\mathbf{W}_{t+1}) - f(\mathbf{W}_t) \leq \eta^2 \|\nabla f(\mathbf{W}_t)\|^2 \left(\frac{L}{2} - \frac{1}{\eta} \right). \quad (70)$$

Choosing $\eta < \frac{2}{L}$

$$f(\mathbf{W}_{t+1}) - f(\mathbf{W}_t) \leq -\gamma \|\nabla f(\mathbf{W}_t)\|^2. \quad (71)$$

Taking a telescoping summation, we have,

$$\sum_{t=0}^{T-1} (f(\mathbf{W}_{t+1}) - f(\mathbf{W}_t)) \leq -\gamma \sum_{t=0}^{T-1} \|\nabla f(\mathbf{W}_t)\|^2 \quad (72)$$

$$\frac{(f(\mathbf{W}_0) - f(\mathbf{W}_{t+1}))}{\gamma} \geq \sum_{t=0}^{T-1} \|\nabla f(\mathbf{W}_t)\|^2. \quad (73)$$

Let us set $\mathbf{W}_{t+1} = \mathbf{W}^*$, where \mathbf{W}^* is a minimizer of f . Then, we have $(f(\mathbf{W}_0) - f(\mathbf{W}_{t+1})) \leq (f(\mathbf{W}_0) - f(\mathbf{W}^*))$, which gives,

$$\frac{(f(\mathbf{W}_0) - f(\mathbf{W}^*))}{\gamma} \geq \sum_{t=0}^{T-1} \|\nabla f(\mathbf{W}_t)\|^2. \quad (74)$$

Dividing both sides by T , we get

$$\frac{(f(\mathbf{W}_0) - f(\mathbf{W}^*))}{\gamma T} \geq \frac{1}{T} \sum_{t=0}^{T-1} \|\nabla f(\mathbf{W}_t)\|^2. \quad (75)$$

If the gradient norm is upper bounded as below, then, trivially we have,

$$\min_{0 \leq t < T} \|\nabla f(\mathbf{W}_t)\|^2 \leq \frac{2(f(\mathbf{W}_0) - f(\mathbf{W}^*))}{\gamma T} \quad (76)$$

This shows that the squared minimum gradient norm goes to zero at a $O(1/T)$ rate, implying convergence to a stationary point in the non-convex sense. \square

Discussion: The result is a direct extension of the convergence guarantees for proximal gradient descent methods on non-convex functions. Proximal gradient methods are widely used in constrained optimization problems, where the update rule involves projecting the parameters onto a feasible set at each iteration. In the case of subspace-constrained networks, the feasible set corresponds to the predefined low-dimensional subspace, and the projection operator ensures that the parameters *partially* remain within this subspace. The convergence rate of $O(1/T)$ in terms of stationarity reflects that, even in the presence of non-convexity and subspace constraints, the optimization algorithm efficiently reduces the gradient norm over time, driving the network parameters toward a stationary point. A possible implication of subspace restriction could be that it acts as an implicit regularizer, encouraging the network to learn more compact and robust representations. This can lead to improved generalization performance on unseen data. We hypothesize that this might be the reason for improved performance of the compressed models over their centralized counterparts (see Fig. 5).

Next result extends the above theorem to the stochastic mini-batch gradient descent optimization.

Theorem H.2. Let $f(\mathbf{W}^{(s)}, \mathbf{W}^{(u)})$ be L -smooth in $\mathbf{W} = (\mathbf{W}^{(s)}, \mathbf{W}^{(u)})$ with $\mathbf{W}^{(s)} \in \mathbf{R}^k$ (constrained block), $\mathbf{W}^{(u)} \in \mathbf{R}^d$ (unconstrained block), and let $\mathcal{S} \subseteq \mathbf{R}^k$ be a closed subspace. Define the feasible set $\mathbf{W} := \mathcal{S} \times \mathbf{R}^d$.

At each iteration t draw a mini-batch and compute an unbiased stochastic gradient $\mathbf{g}_t = (\mathbf{g}_t^{(s)}, \mathbf{g}_t^{(u)})$ satisfying

$$\mathbb{E}[\mathbf{g}_t | \mathbf{W}_t] = \nabla f(\mathbf{W}_t), \quad \mathbb{E}[\|\mathbf{g}_t - \nabla f(\mathbf{W}_t)\|^2 | \mathbf{W}_t] \leq \sigma^2 \quad (\text{bounded variance}). \quad (77)$$

Update the parameters with the stochastic partially-projected gradient (SPPG) step

$$\begin{aligned} \widetilde{\mathbf{W}}_{t+1}^{(s)} &= \mathbf{W}_t^{(s)} - \eta_t \mathbf{g}_t^{(s)}, & \widetilde{\mathbf{W}}_{t+1}^{(u)} &= \mathbf{W}_t^{(u)} - \eta_t \mathbf{g}_t^{(u)}, \\ \mathbf{W}_{t+1}^{(s)} &= \Pi_{\mathcal{S}}(\widetilde{\mathbf{W}}_{t+1}^{(s)}), & \mathbf{W}_{t+1}^{(u)} &= \widetilde{\mathbf{W}}_{t+1}^{(u)}, \end{aligned}$$

where each stepsize satisfies $\eta_t \in (0, 1/L]$.

Let $f^* := \min_{\mathbf{W} \in \mathbf{W}} f(\mathbf{W})$ and denote $\bar{\eta} := \frac{1}{T} \sum_{t=0}^{T-1} \eta_t$. Then for every horizon $T \geq 1$

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\mathbf{W}_t)\|^2] \leq \frac{2(f(\mathbf{W}_0) - f^*)}{\bar{\eta} T} + L \bar{\eta} \sigma^2. \quad (78)$$

In particular, with the constant stepsize $\eta_t \equiv \eta \leq 1/L$ it holds that

$$\min_{0 \leq t < T} \mathbb{E}[\|\nabla f(\mathbf{W}_t)\|^2] \leq \frac{2(f(\mathbf{W}_0) - f^*)}{\eta T} + L \eta \sigma^2,$$

which attains the optimal $\mathcal{O}(1/\sqrt{T})$ rate when $\eta = \min\{1/L, \sqrt{2(f(\mathbf{W}_0) - f^*)/(L\sigma^2 T)}\}$.

Proof. The argument follows the deterministic proof but replaces exact gradients with the stochastic estimator and takes conditional expectations at the appropriate points.

Because the Euclidean projection onto a closed subspace is 1-Lipschitz, for every t $\|\mathbf{W}_{t+1}^{(s)} - \mathbf{W}_t^{(s)}\| \leq \|\widetilde{\mathbf{W}}_{t+1}^{(s)} - \mathbf{W}_t^{(s)}\| = \eta_t \|\mathbf{g}_t^{(s)}\|$. The unconstrained coordinates remain unprojected, hence $\|\mathbf{W}_{t+1}^{(u)} - \mathbf{W}_t^{(u)}\| = \eta_t \|\mathbf{g}_t^{(u)}\|$. Using the Pythagorean theorem for the two disjoint blocks,

$$\|\mathbf{W}_{t+1} - \mathbf{W}_t\| \leq \eta_t \|\mathbf{g}_t\|. \quad (79)$$

L -smoothness of f implies

$$f(\mathbf{W}_{t+1}) \leq f(\mathbf{W}_t) + \nabla f(\mathbf{W}_t)^\top (\mathbf{W}_{t+1} - \mathbf{W}_t) + \frac{L}{2} \|\mathbf{W}_{t+1} - \mathbf{W}_t\|^2.$$

Introduce the zero-mean “noise” term $\epsilon_t := \mathbf{g}_t - \nabla f(\mathbf{W}_t)$. Conditioned on \mathbf{W}_t , $\mathbb{E}[\epsilon_t] = 0$ and $\mathbb{E}[\|\epsilon_t\|^2] \leq \sigma^2$. Using (79) and $\mathbf{g}_t = \nabla f(\mathbf{W}_t) + \epsilon_t$,

$$\begin{aligned} \mathbb{E}[f(\mathbf{W}_{t+1}) \mid \mathbf{W}_t] &\leq f(\mathbf{W}_t) - \eta_t \|\nabla f(\mathbf{W}_t)\|^2 + \frac{L\eta_t^2}{2} \mathbb{E}[\|\mathbf{g}_t\|^2 \mid \mathbf{W}_t] \\ &\leq f(\mathbf{W}_t) - \eta_t \left(1 - \frac{L\eta_t}{2}\right) \|\nabla f(\mathbf{W}_t)\|^2 + \frac{L\eta_t^2}{2} \sigma^2, \end{aligned}$$

where the last line used $\mathbb{E}[\|\mathbf{g}_t\|^2] = \|\nabla f(\mathbf{W}_t)\|^2 + \mathbb{E}[\|\epsilon_t\|^2] \leq \|\nabla f(\mathbf{W}_t)\|^2 + \sigma^2$. Because $\eta_t \leq 1/L$, $1 - L\eta_t/2 \geq 1/2$. Hence

$$\mathbb{E}[f(\mathbf{W}_{t+1})] \leq \mathbb{E}[f(\mathbf{W}_t)] - \frac{\eta_t}{2} \mathbb{E}[\|\nabla f(\mathbf{W}_t)\|^2] + \frac{L\eta_t^2}{2} \sigma^2. \quad (80)$$

Sum (80) over $t = 0, \dots, T-1$ and rearrange:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\mathbf{W}_t)\|^2] \leq \frac{2(\mathbb{E}[f(\mathbf{W}_0)] - f^*)}{\sum_{t=0}^{T-1} \eta_t} + L\sigma^2 \frac{\sum_{t=0}^{T-1} \eta_t^2}{\sum_{t=0}^{T-1} \eta_t}.$$

Noting that $\sum_t \eta_t = T\bar{\eta}$ and $\sum_t \eta_t^2 \leq T\bar{\eta}^2$ (by Jensen’s inequality) yields

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\|\nabla f(\mathbf{W}_t)\|^2] \leq \frac{2(f(\mathbf{W}_0) - f^*)}{\bar{\eta}T} + L\bar{\eta}\sigma^2,$$

which is precisely (78). □

I Ablations

To validate the utility of the proposed method, we stress-test it with different network and training configurations. All the models are trained on C4.

I.1 Comparison at different bandwidth thresholds

We observe that top-k sparsification allows up to a $10\times$ compression and 8-bit quantization enables around $4\times$ compression before significant degradation in convergence. Accordingly, we fix those compression ratios for the baselines. We exclude SVD-based low-rank method as it is extremely slow.

Our method achieves up to $100\times$ compression without compromising convergence. This allows it to outperform other methods even when the available bandwidth drops to 80Mbps. Table 5 show that our approach demonstrates a consistent TPS compared to other methods across the range of bandwidths, making it highly practical for pipeline-parallel training over internet-grade connections.

I.2 Applying compression on Swarm

We further conducted an experiment with Swarm parallelism [40]. Swarm’s primary goals are fault tolerance and dynamic volunteer participation rather than communication efficiency. Swarm implementation relies on standard activation and gradient quantization for compression. Our technique is therefore orthogonal to swarm too and can be layered on top of Swarm to further reduce bandwidth. To demonstrate this, we conducted an additional experiment with a 600M parameter model that applies our compression within a Swarm setting; the results are provided in Table 6.

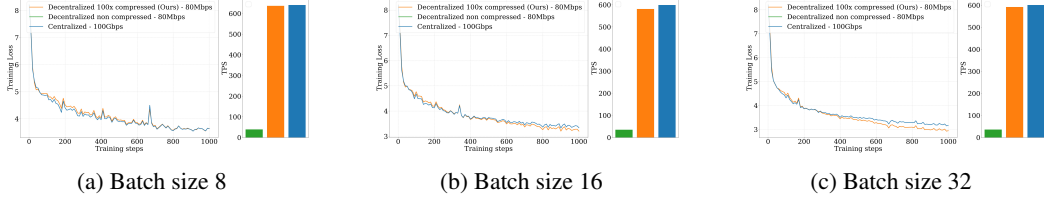


Figure 8: **Convergence with different batch sizes.** Each block shows the training curves (with respect to training steps) and throughput for an 8-layer, 2B-parameter model on C4. Decentralized configurations use 80 Mbps connections, while the centralized configuration uses datacenter-grade 100 Gbps links. Even under a 80 Mbps bandwidth budget, our compressed model achieves convergence comparable to—and sometimes exceeding—the centralized configuration across varying batch sizes. Note that as the batch size increases, the compressed model achieves increasingly better results compared to the centralized model. The iteration-wise dynamics of the uncompressed decentralized model match those of the centralized model; hence, we omit its curve for clarity. Despite severe bandwidth constraints, our compressed model attains throughput on par with the centralized setting. In contrast, the uncompressed decentralized setup suffers significantly lower throughput due to communication bottlenecks. All models share the same network architecture.

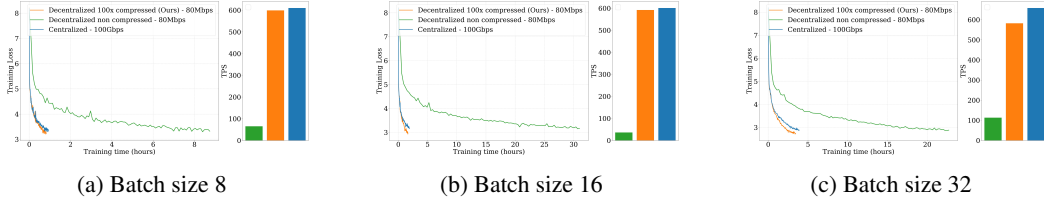


Figure 9: **Convergence with different batch sizes.** Each block shows the training curves (with respect to wall-clock time) and throughput for an 8-layer, 2B-parameter model on C4. Decentralized configurations use 80 Mbps connections, while the centralized configuration uses datacenter-grade 100 Gbps links. Even under a 80 Mbps bandwidth budget, our compressed model achieves convergence comparable to—and sometimes exceeding—the centralized configuration across varying batch sizes while the decentralized non compressed model demonstrates extremely slow convergence.

J Analysis of pretrained checkpoints

We investigate the stable ranks of the output projection layers of the official checkpoints of frontier open-weight LLMs (LlaMA, Qwen, Olmo, Phi). The observations are reported in Fig. 16. As shown, the weights demonstrate extremely low ranks across all the layers and models.

K Memory Overhead Analysis

As our method requires addition of fixed high-rank and dynamic low-rank embeddings, one potential concern is the memory overhead, as sequence length L grows. We empirically and theoretically demonstrate that the absolute memory overhead introduced by our approach remains constant and negligible, while the relative overhead decreases with increasing sequence lengths.

We first empirically validate this and then give a theoretical explanation. To this end, we performed experiments using a 2B-parameter Transformer model (8 layers, 4k model dimension, 16 attention heads) distributed across eight NVIDIA H100 GPUs under varying sequence lengths L . Table 7 summarizes these results. Remarkably, the absolute overhead consistently remains around 400 MB, irrespective of sequence length. As L grows from 8k to 24k, the relative memory overhead correspondingly drops from 4.0% to 0.6%.

This consistent and negligible overhead can be explained through PyTorch’s memory management behavior. Firstly, fixed embedding lookups in our method are ephemeral since they are non-trainable; PyTorch does not store activation gradients for them. After they are added to the activations, PyTorch’s caching allocator reuses the memory (instead of invoking `cudaMalloc/cudaFree`), allowing temporary

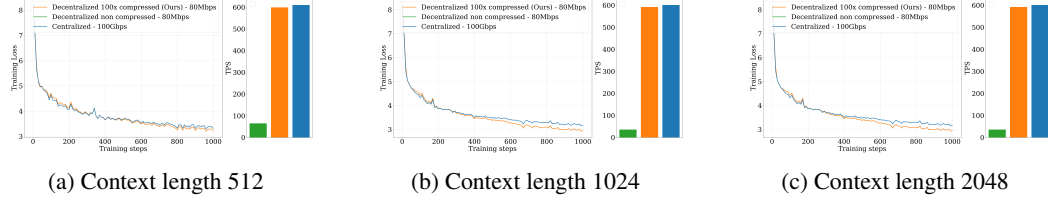


Figure 10: Convergence with different context lengths. Each block shows the training curves (with respect to training steps) and throughput for an 8-layer, 2B-parameter model on C4. Decentralized configurations use 80 Mbps connections, while the centralized configuration uses datacenter-grade 100 Gbps links. Even under a 80 Mbps bandwidth budget, our compressed model achieves convergence comparable to—and sometimes exceeding—the centralized configuration across varying context lengths. Note that as the context length increases, the compressed model achieves increasingly better results compared to the centralized model. The iteration-wise dynamics of the uncompressed decentralized model match those of the centralized model; hence, we omit its curve for clarity. Despite severe bandwidth constraints, our compressed model attains throughput on par with the centralized setting. In contrast, the uncompressed decentralized setup suffers significantly lower throughput due to communication bottlenecks.

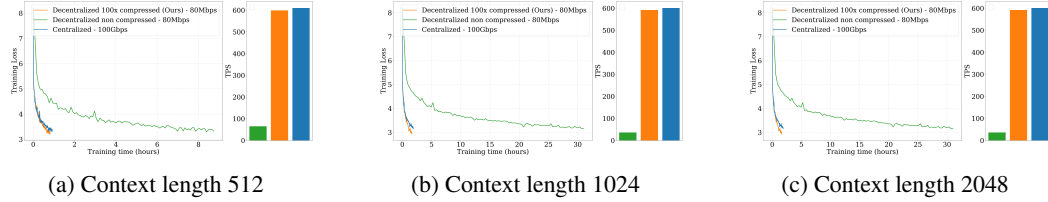


Figure 11: Convergence with different context lengths. Each block shows the training curves (with respect to wall-clock time) and throughput for an 8-layer, 2B-parameter model on C4. Decentralized configurations use 80 Mbps connections, while the centralized configuration uses datacenter-grade 100 Gbps links. Even under a 80 Mbps bandwidth budget, our compressed model achieves convergence comparable to—and sometimes exceeding—the centralized configuration across varying context lengths while the decentralized non compressed model demonstrates extremely slow convergence.

tensors to be released prior to attention. Thus, their memory footprint doesn’t persist into later stages. Secondly, embedding lookups ($\mathcal{O}(B \times L \times D)$ memory usage) are inherently minor relative to attention ($\mathcal{O}(B \times L^2 \times D)$) and MLP layers ($\mathcal{O}(B \times L \times D^2)$), making their temporary storage impact negligible in peak memory usage.

Further, the cached embedding tables themselves do not pose a meaningful threat to scalability. For instance, even a large embedding table (10k-dimensional embedding and 50k vocabulary size) would consume approximately 0.93 GB in 16-bit precision, requiring only impractically large embedding dimensions (e.g., 100k+) to surpass 10 GB. State-of-the-art frontier models (e.g., DeepSeek [8], LLAMA-405B [14]) remain far below such thresholds (7K and 16K, respectively). Therefore, even with significantly increased embedding dimensions, memory overhead remains dominated by MLP and attention computations, not embedding storage.

In conclusion, our method demonstrates robust and scalable memory behavior, effectively managing memory overhead even at extremely large sequence lengths, validating its practicality for decentralized training at scale.

L Memory Overhead with Increasing Workers

Another key consideration is how our compression scheme scales in terms of memory overhead with an increasing number of workers. One might suspect that distributing long contexts across more workers could lead to increased overhead per worker. We clarify that this concern is unfounded due to the way embeddings and attention computations are handled in our setup.

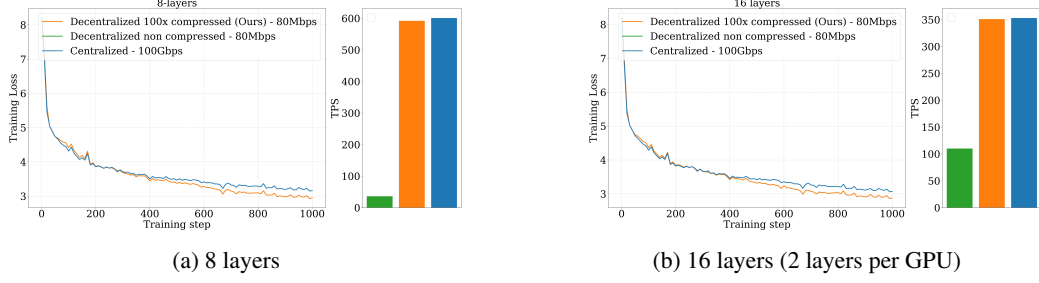


Figure 12: **Convergence with increasing number of layers.** Note that as the number of layer increase, our model consistently matches (even slightly exceeds) the centralized model. This is in stark contrast to lossy compression schemes, where the model convergence severely degrades as the model depth increases [6].

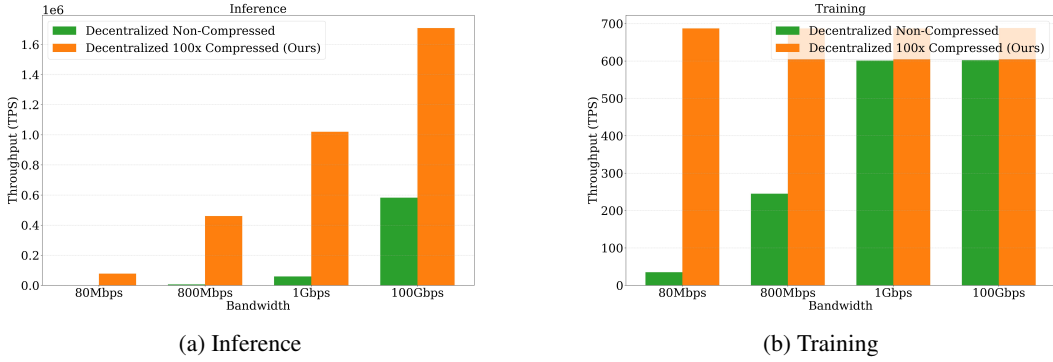


Figure 13: **Throughput across bandwidth constraints.** We limit network bandwidth between GPUs and measure throughput during inference and training. The compressed model consistently achieves significantly higher throughput than the non-compressed model. Notably, even at 100Gbps, compression improves inference throughput by 3 \times , demonstrating benefits for even centralized setups.

To elaborate, in a context-parallel training setting, each worker handles a distinct segment of the input sequence. Embedding lookups are performed *locally on each worker* before attention computations. The resulting key-value (KV) tensors maintain the same shape and size as a standard, non-compressed model, ensuring no extra storage requirement. Importantly, the ephemeral embedding additions are discarded immediately after their local usage, well before the memory-intensive attention and MLP layers, thus never contributing to peak memory usage.

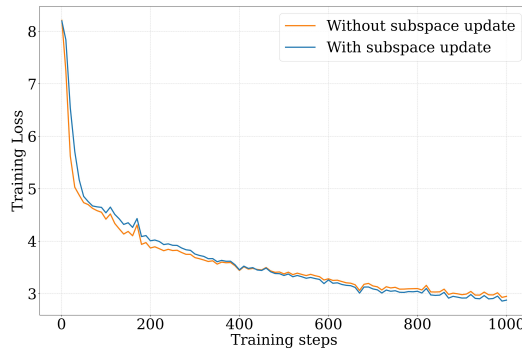


Figure 14: **Effect of the (Grassman) subspace updates.** An 8-layer model is trained on C4 for this experiment. Since this performance gap seems to keep increasing towards the end of training, we emphasize the importance of infrequent subspace updates.

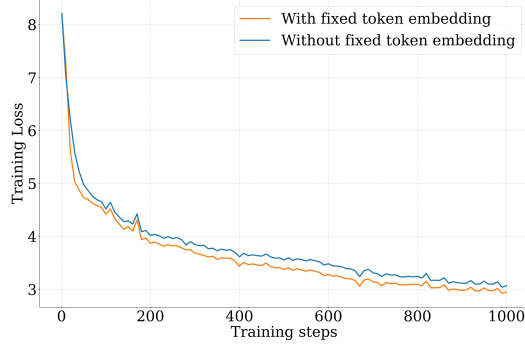


Figure 15: **Effect of fixed token embedding.** We decompose the token embeddings to a fixed high rank embedding and a dynamic low rank embedding. As shown in the figure, we observe inferior convergence when there is no such decomposition. An 8-layer model is trained on C4 for this experiment.

Method	TPS @ 100 Gbps	TPS @ 1 Gbps	TPS @ 800 Mbps	TPS @ 500 Mbps	TPS @ 80 Mbps
Baseline	602	378	333	222	36
Top-K	612	532	521	451	129
Quantization	608	499	487	415	83
Ours	624	612	610	603	592

Table 5: Throughput (TPS) comparison under varying bandwidths.

To empirically validate our claims, we extend our experiments to include varying numbers of workers using Ring Attention, a state-of-the-art context-parallel mechanism where each worker exchanges KV tensors only with its immediate neighbors. As embedding operations occur locally and KV tensors remain unchanged in size, our compression scheme integrates seamlessly.

Table 8 illustrates that, irrespective of the increase in sequence length (L) and corresponding increase in the number of workers, our per-worker memory overhead remains constant (around 400MB) and does not scale with either sequence length or worker count.

Thus, our design effectively maintains low, constant memory overhead per worker, demonstrating excellent scalability in decentralized contexts. The embedding memory per-worker overhead remains negligible and constant, independent of the scaling of sequence length or the number of distributed workers, validating our approach for large-scale decentralized training scenarios.

L.1 Computational Overhead of subspace updates

Our proposed method introduces two additional computational components: *weight projection* and *subspace updates*. Here, we provide an analysis of the overhead associated with these operations, demonstrating their minimal impact on memory usage and computational efficiency.

Overhead of Weight Projection. We first assess the computational cost incurred by the weight projection step. For a practical evaluation, we consider a model with approximately 2 billion parameters (8 layers, 4k model dimension, 16 attention heads), pipelined across 8 A10G GPUs. In our experiments, a single forward pass takes approximately 4.61s, while the weight projection step adds only about 0.05s of computation time. Thus, the relative overhead introduced by weight projection is approximately 1%, indicating a negligible computational burden.

Table 6: **Applying our compression on Swarm.**

Method	Loss	TPS
Swarm	6.05	1,502
Swarm + our compression	4.28	17,072

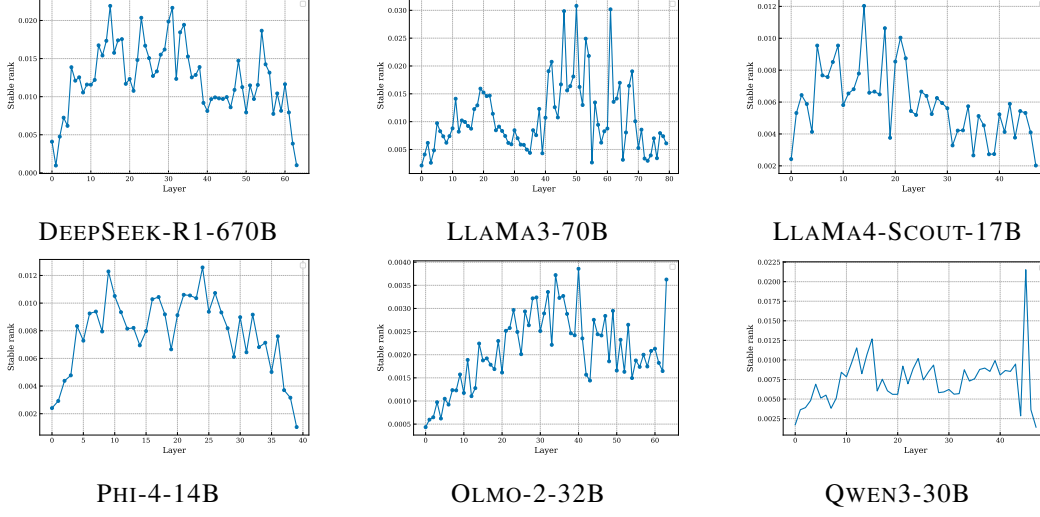


Figure 16: **Stable ranks of output projection matrices (normalized by the maximum possible rank) across different frontier models.** Statistics are computed on official fully pre-trained checkpoints. All the models demonstrate extremely low ranks, solidifying our theoretical arguments.

Table 7: Peak memory usage comparison between baseline and our subspace method as sequence length (L) scales.

L	Baseline (GB)	Ours (GB)	Overhead	Relative Overhead
8k	9.66	10.06	$\sim 400\text{MB}$	$\sim 4.0\%$
16k	36.51	36.91	$\sim 400\text{MB}$	$\sim 1.1\%$
24k	76.00	76.46	$\sim 400\text{MB}$	$\sim 0.6\%$

Overhead of Subspace Updates. Updating the subspace via Grassmann manifold optimization initially appears complex; however, by deriving closed-form Euclidean gradients, the practical implementation becomes computationally inexpensive. Specifically, given a Grassmann loss defined as:

$$\mathcal{L}_{\text{Grassmann}} = \frac{1}{K} \sum_{t=k}^{k+K} \|\nabla_L(\mathbf{X}_t^{\text{final}}) (\mathbf{I} - \mathbf{U}_k \mathbf{U}_k^\top)\|_F^2,$$

and using the Frobenius norm expansion, we obtain:

$$\mathcal{L}_{\text{Grassmann}} = \text{const} - \frac{1}{K} \sum_t \text{Tr}(\mathbf{G}_t \mathbf{U}_k \mathbf{U}_k^\top \mathbf{G}_t^\top),$$

where $\mathbf{G}_t = \nabla_L(\mathbf{X}_t^{\text{final}})$. By defining the symmetric accumulation matrix $\mathbf{S} = \frac{1}{K} \sum_t \mathbf{G}_t^\top \mathbf{G}_t$, the closed-form Euclidean gradient simplifies elegantly to:

$$\nabla_{\mathcal{L}_{\text{Grassmann}}}(\mathbf{U}_k) = -2\mathbf{S}\mathbf{U}_k.$$

Practically, implementing this gradient involves straightforward matrix operations in PyTorch:

- **Accumulation:** Compute and accumulate $\mathbf{G}_t^\top \mathbf{G}_t$ once per iteration via standard matrix multiplication.
- **Gradient Computation:** Multiply \mathbf{S} by \mathbf{U}_k once per a fixed interval (e.g., every 500 iterations).
- **Riemannian Projection:** Project back onto the Grassmann manifold via a basic linear algebra operation, again executed at low frequency.

These matrix operations are inexpensive and infrequent, resulting in minimal computational overhead. Consequently, our method’s overall cost of maintaining and updating the subspace is negligible

Table 8: Peak memory per worker with increasing sequence lengths and workers using our compression scheme.

L	Num. Workers	Baseline (GB)	Ours (GB)	Overhead/Worker	Relative Overhead
8k	1	9.66	10.06	$\sim 400\text{MB}$	$\sim 4.0\%$
16k	1	36.51	36.91	$\sim 400\text{MB}$	$\sim 1.1\%$
24k	1	76.00	76.46	$\sim 400\text{MB}$	$\sim 0.6\%$
50k	2	76.13	76.55	$\sim 400\text{MB}$	$\sim 0.55\%$
65k	3	78.19	79.62	$\sim 400\text{MB}$	$\sim 0.54\%$

compared to the cost of the forward and backward passes of transformer models, affirming the efficiency of our approach.