LOCAL PROXIES FOR MEMORY-EFFICIENT TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Local learning algorithms can reshape resource requirements of artificial intelligence. The memory bottleneck of standard end-to-end backpropagation lies in the need to store all network activations until the loss gradient has been propagated back through. Local learning avoids this, but typically at a cost in accuracy. We study how to achieve accurate and memory efficient local learning in transformers. For each transformer block, we train a local proxy network to predict the global network output via distillation. Gradients can be independently propagated through this local proxy while estimating well the backpropagation gradients. In theory, this approach is exact for deep linear networks; in non-linear residual models, the gradient error can be bounded when using the appropriate residual proxy. We verify this numerically and demonstrate that optimizing the distillation loss improves gradient estimation. Evaluating the performance of proxy propagation on CIFAR-100 in vision transformers, we find that it achieves a better accuracy/memory trade-off compared to other local algorithms. Finally, we show that the method is further relevant for fine-tuning large language models on a small memory budget. Our method trains faster and achieves higher performance than zeroth-order memory-efficient learning algorithms with comparable memory cost.

1 INTRODUCTION

Conventionally, intelligence emerges in neural network models from end-to-end training with backpropagation (Rumelhart et al., 1986). By design, backpropagation induces a backward locking problem (Jaderberg et al., 2017). A forward pass first computes the network activations from inputs to outputs and then the backward pass propagates gradients in reverse order. In the meantime, network activations need to be stored, increasing memory requirements with network depth. The large memory consumption is a major manufacturing cost and increases the energy consumption of training datacenters (Gholami et al., 2024). When training large language models (LLMs) under a limited memory budget, the network depth and sequence length that fit in memory therefore constrain the range of tasks and the reasoning capability of the model.

Historically inspired by neuroscience research (Hebb, 1949; Markram et al., 2011; Gerstner et al., 2018), local learning is one way of avoiding the backward locking problem. Since the brain appears to be a distributed network without a precise backward gradient communication pathway, it is natural to suppose that biological sub-networks mainly learn from their immediate inputs and internal representations (Lillicrap et al., 2020). Yet few local learning algorithms have been scaled to transformers, and attempts so far have had limited success (Filipovich et al., 2022). Most of these algorithms were not designed to approximate the backpropagated gradient, so they struggle in deep architectures; on top of this, those designed for biological plausibility often come with an increase in training time. Within the scope of local learning, we seek an alternative to backpropagation that is mathematically grounded and memory efficient without compromising training time.

Towards this goal, we define locality by partitioning a large transformer into blocks, with the gradient computation in each block decoupled in the sense of Jaderberg et al. (2017): block parameters are updated immediately after the forward pass of the block. Hence, we say that learning is "local" if it needs to store activations from only one block at a time in memory. Additionally, when the block updates are fully independent, we say that the training algorithm is "unlocked", making possible an efficient form of model parallelism (Belilovsky et al., 2020). Here, we follow a principled approach to estimate global gradients locally. To each block we equip a local proxy network with the objective of predicting the global network output via distillation. Local gradients are then computed with local

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.

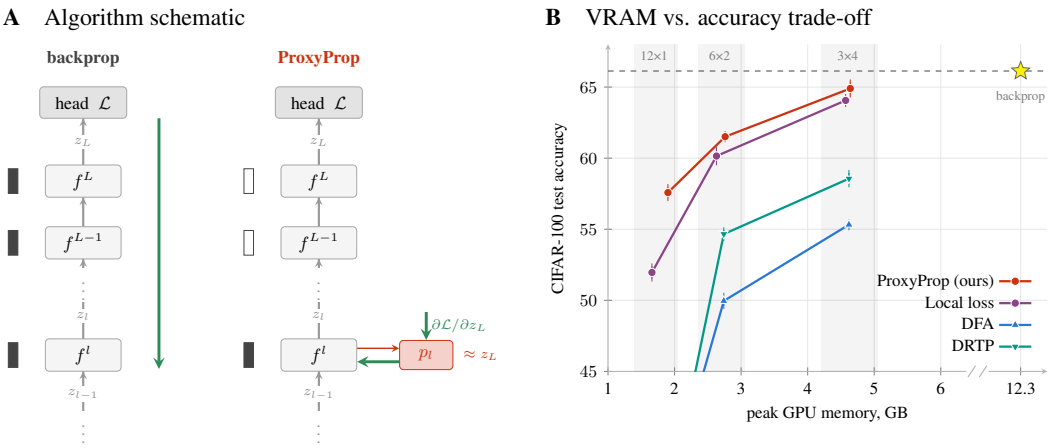

Figure 1: **ProxyProp and its accuracy/memory trade-off.** **1A**: In ProxyProp, when updating block f^ℓ, only activations of that block need to be stored. The loss gradients are propagated through a proxy function instead of the other blocks. **1B**: Memory and accuracy tradeoff for ProxyProp and other local methods; the 12-layer vision transformer (ViT) is either partitioned into 3, 6 or 12 blocks.

backpropagation through the proxy instead of the downstream blocks. This way, proxy propagation (ProxyProp) offers theoretical insights that translate into a practical algorithm:

- *Theory:* ProxyProp is an approximation of the global backpropagation. Despite the locality and the simplicity of the prediction objective, a linear proxy implicitly estimates the expected network Jacobian, and is therefore exact in deep linear networks.
- *Theory:* For non-linear residual networks, when the proxy follows a residual parameterization, the local gradients are exact at initialization and can achieve a bounded gradient estimation error when the skip connections of the network are dominating.
- *Simulation:* For supervised classification in vision transformers (ViT), across various network partitionings, ProxyProp offers a better trade-off between memory and accuracy over other local algorithms.
- *Simulation:* ProxyProp enables memory-efficient fine-tuning of LLMs, achieving higher performance and faster training than zeroth-order optimization techniques (MeZO) at comparable memory cost.

2 Review: State of the Art

The "local learning" terminology has been defined with different aims in different contexts (see Bredenberg et al. (2023) for a formal review). In bio-inspired computing, it is a constraint for bio-plausible learning circuitry. This line of research originated from the observation that backpropagation requires a backward circuit with symmetric feedback weights, for which there is no anatomical evidence (Lillicrap et al., 2020). Different lines of work have relaxed this requirement (Lillicrap et al., 2016; Akrout et al., 2019; Max et al., 2024; Nøkland, 2016), or sought compatibility with Hebbian learning or predictive coding (Whittington & Bogacz, 2017; Scellier & Bengio, 2017; Halvagal & Zenke, 2023; Illing et al., 2021). All these algorithms formalize how to replace backpropagation through the network hierarchy in space, which is complementary to the replacement of the backpropagation of information through time (Bellec et al., 2020; Murray, 2019). Together, these algorithms were foundational in bringing local learning theory to deep networks and have inspired circuitry for learning in physical systems (Momeni et al., 2023), photonics (Wang et al., 2026), analog hardware (Pehle et al., 2022) and digital ASIC technology (Su & Indiveri, 2025).

We focus in particular on decoupling the stack of network layers into L gradient-isolated blocks, where each block learns locally. This can be achieved by stopping the gradient propagation across blocks and optimizing each network block with a local loss, either being supervised

(Belilovsky et al., 2019; Nøkland & Eidnes, 2019; Frenkel et al., 2021; Shing et al., 2026) or self-supervised (Löwe et al., 2019; Illing et al., 2021; Hinton, 2022; Halvagal & Zenke, 2023; Siddiqui et al., 2024; Chen et al., 2025). Such methods have also been applied to finetuning pretrained models (Key et al., 2023; Gelbard et al., 2026). Although local losses work relatively well in practice, it is not understood how the stack of decoupled blocks builds a hierarchy of representations (see Delrocq et al. (2026); Korchinski et al. (2026) for some analyses).

Towards a principled algorithm to approximate the global gradient from local information, we build instead a gradient estimate $g^{loc}_{\theta_\ell}$ for layer ℓ parameters θ_ℓ that intends to approximate the backpropagation gradient $\partial\mathcal{L}/\partial\theta_\ell$ denoted $g^{BP}_{\theta_\ell}$. The difficulty is that $g^{loc}_{\theta_\ell}$ needs to estimate a global quantity from local network activations. There are only few local theories explaining the emergence of globally coherent gradient from local calculations. For a class of self-supervised contrastive losses (Zihan et al., 2026), the local estimations can be exact, shown to work well to train unsupervised convolutional networks. Alternatively, Jaderberg et al. (2017) train a local auxiliary network to explicitly predict the gradient bootstrapped from the following block. However, in practice, the compounding errors of the auxiliary networks make it numerically challenging.

3 Theory: proxy distillation with accurate gradient propagation

Definition of ProxyProp. We partition a deep network into L blocks such that: $z_\ell = f^\ell_\theta(z_{\ell-1})$ where $z_\ell \in \mathbb{R}^{d_\ell}$ is the activation of block ℓ and z_0 is the input. We consider an arbitrary loss head $\mathcal{L}(\theta_h, z_L, y)$ that typically includes head parameters θ_h and may depend on target tokens or labels y. For brevity, we write $\mathcal{L}(z_L)$. The aim is to approximate the backpropagated gradients $g^{BP}_{\theta_\ell}$.

Our method defines a proxy network p^ℓ_ϕ distilled from the network output z_L such that $z_L \approx p^\ell_\phi(z_\ell)$. For memory efficiency, we construct the proxy such that $p^\ell_\phi(z_\ell)$ or its gradient propagation (*i.e.* $\partial p^\ell_\phi/\partial z_\ell$) does not require large memory storage. Typically, $p^\ell_\phi(z_\ell)$ can be either a linear function of z_ℓ or a tiny transformer. Under this construction summarized in Figure 1, our algorithm ProxyProp backpropagates the loss gradients through the proxy instead of the expensive layers f_θ such that our estimator of the full backpropagation gradients can be written as:

$$g^{loc}_\theta = \frac{\partial f^\ell_\theta}{\partial \theta}^\top \frac{\partial p^\ell_\phi}{\partial z_\ell}^\top \frac{\partial \mathcal{L}}{\partial z_L} \quad \text{(ProxyProp)} \tag{1}$$

This equation is the core of ProxyProp (Algorithm 1). The algorithm is "local": after the calculation of f^ℓ_θ (line 7 of Algorithm 1), $g^{loc}_{\theta_\ell}$ can be computed and the parameters of block ℓ can be updated at once, without waiting for the forward or backward pass of the later blocks. Consequently, we can release the stored activation $z_{\ell-1}$ (line 11) immediately after, such that the memory requirement for network activations does not scale with the network depth L as it would in backpropagation. Table A1 in appendix summarizes this: while the memory requirement scales linearly with depth in backpropagation, it is independent of the depth in ProxyProp.

We make three further clarifications *(a-c)* relevant to understanding the design of this algorithm. *(a)* Algorithm 1 is "local", and the weight updates are "unlocked" such that (line 6 to 11) could be parallelized. *(b)* The right hand term $\partial\mathcal{L}/\partial z_L$ requires a *pre*-forward pass (line 1 to 5) to compute the last network activation z_L and the loss gradient. Thus, there remains a weaker "locking" at this level. In the discussion we describe a ProxyProp-fully-unlocked variant (Algorithm 2) which can be fully distributed. *(c)* In this paper, we study ProxyProp as described in Algorithm 1, our theoretical analyses relying on the observation that g^{loc}_θ becomes exact if the proxy Jacobian equals the large network Jacobian: $\partial p^\ell_\phi/\partial z_\ell = \partial z_L/\partial z_\ell$.

Proxy distillation objective. Notoriously in the synthetic gradient literature (Jaderberg et al., 2017), the difficulty in efficient decoupled neural interface (DNI) lies in approximating well the backpropagated gradients at each block. Our strategy distills the large network into the proxy p_ϕ, minimizing:

$$\phi^* = \text{argmin}_\phi \, ||z_L - p_\phi(z_\ell)||^2 \tag{2}$$

Distillation from the large network output z_L does not guarantee *a priori* that the backpropagated gradient or the Jacobians $\partial p^\ell_\phi/\partial z_\ell$ and $\partial z_L/\partial z_\ell$ will become equal, which is necessary to achieve accurate gradients according to clarification *c*. Alternatives include a bootstrapped synthetic gradient loss (Jaderberg et al., 2017) and explicit Jacobian matching (Czarnecki et al., 2017); in our

Algorithm 1: Description of the ProxyProp algorithm. No activations or gradients are stored or computed unless specified. The input z_0 is given. $p_\phi^L(z_l)$ is always the identity.

```
/* Pre-forward pass and loss head update */
for ℓ = 1 to L do
    z_ℓ ← f_ℓ(z_{ℓ−1});
    delete z_{ℓ−1};
Store z_L and ∂L/∂z_L;
Update θ_h with ∂L/∂θ_h ;
/* Local ProxyProp algorithm */
for ℓ = 1 to L do
    z_ℓ ← f_ℓ(z_{ℓ−1});
    Compute the proxy p^ℓ_φ(z_ℓ);
    Update θ_ℓ with local gradient from Equation 1;
    Update φ with gradient step on the distillation loss from Equation 2;
    delete z_{ℓ−1};
```

experiments in Section 4, these either gave inaccurate gradients or required second-order derivatives, resulting in substantial memory consumption. By extension of the results from Srinivas & Fleuret (2018), if Equation 2 is minimized under random perturbations of z_ℓ, then so, implicitly, is the squared error of the Jacobians. In practice, we even observed that minimizing Equation 2 aligns the backpropagated gradients without repeated perturbations. Below, we suggest an alternative theory that is well supported by our simulations.

Gradient consistency in linear proxies. We may first consider a proxy function that is linear. This case allows for a theoretical treatment explaining cases where optimizing Equation 2 yields accurate gradients. We consider a linear proxy function $p_\phi(z_\ell) = Pz_\ell + b$ with parameters $\phi = (P, b)$ that predict the output layer z_L using the hidden layer z_ℓ. For each iteration and parameter θ, we can resolve in closed form the parameters $\phi^\star = (P_\ell^*, b_\ell^*)$ (minimizing Equation 2) and analyze the resulting gradients. In this setting, we find a theoretical link between the Jacobian of the proxy $\partial p_\phi^\ell / \partial z_\ell = P_l^*$ and the unknown Jacobian $\partial z_L / \partial z_\ell$ of the large network:

Proposition 3.1. *For any (non-linear) network distilled with a linear proxy $p_\phi(z_\ell) = Pz_\ell + b$, if z_ℓ follows a Gaussian distribution $\mathcal{N}(\mu_\ell, \Sigma_\ell)$ with Σ_ℓ positive definite, and the layers $f_{\ell'}$ ($\ell' > \ell$) are locally Lipschitz such that $\mathbb{E}\left[\left\|\partial z_L / \partial z_\ell\right\|\right] < \infty$, the matrix P_ℓ^* of the optimal linear proxy is the expected Jacobian of the remaining layer stack:*

$$P_\ell^* = \mathbb{E}_{z_\ell}\left[\frac{\partial z_L}{\partial z_\ell}\right] \tag{3}$$

Proof. Let's first resolve the closed-form solution for the parameters of the optimal linear proxy: $(P_\ell^*, b_\ell^*) = \arg\min_{P,b} \mathbb{E}\left\| z_L - Pz_\ell - b\right\|^2$. Setting the derivative with respect to b_ℓ to zero gives $b_\ell^\star = \mathbb{E}[z_L] - P\mathbb{E}[z_\ell]$, and substituting back, only the centered features remain, leaving us to minimize $\mathbb{E}\|\bar{z}_L - P\bar{z}_\ell\|^2$ where $\bar{z}_\ell = z_\ell - \mathbb{E}[z_\ell]$. Setting the derivative with respect to P to zero gives $P_\ell^* = \mathrm{Cov}(\bar{z}_L, \bar{z}_\ell)\,\Sigma_\ell^{-1}$, where the positive definiteness of Σ_ℓ guarantees the existence of the inverse. Then we define $F_\ell(z_\ell) = f_L \circ f_{L-1} \cdots \circ f_{\ell+1}(z_\ell)$ which is also locally Lipschitz with $\mathbb{E}\|\partial z_L / \partial z_\ell\| < \infty$, and since z_ℓ is Gaussian with covariance Σ_ℓ, the multivariate Stein's Lemma gives $\mathbb{E}[z_L \bar{z}_\ell^\top] = \mathbb{E}[\frac{\partial z_L}{\partial z_\ell}]\Sigma_\ell$. On the other hand, $\mathrm{Cov}(\bar{z}_L, \bar{z}_\ell) = \mathbb{E}[(z_L - \mathbb{E}[z_L])\bar{z}_\ell^\top] = \mathbb{E}[z_L \bar{z}_\ell^\top] - \mathbb{E}[z_L]\mathbb{E}[\bar{z}_\ell]^\top = \mathbb{E}[z_L \bar{z}_\ell^\top]$ because $\mathbb{E}[\bar{z}_\ell] = 0$. Therefore, we can replace the outer product of Stein's Lemma to get: $\mathbb{E}[\frac{\partial z_L}{\partial z_\ell}] = \mathbb{E}[z_L \bar{z}_\ell^\top]\Sigma_\ell^{-1} = \mathrm{Cov}(\bar{z}_L, \bar{z}_\ell)\Sigma_\ell^{-1} = P_\ell^*$. □

Corollary 3.2. *If the input z_0 is Gaussian with positive definite covariance Σ_0, and F_ℓ is a deep linear network (f_θ^ℓ linear for all ℓ) with full-row-rank feedforward weights, ProxyProp (Equation 1) with linear proxy ϕ^* implements exactly the same algorithm as backpropagation:*

$$\textit{(ProxyProp)}\ g_\theta^{loc} = g_\theta^{BP}\ \textit{(backpropagation)} \tag{4}$$

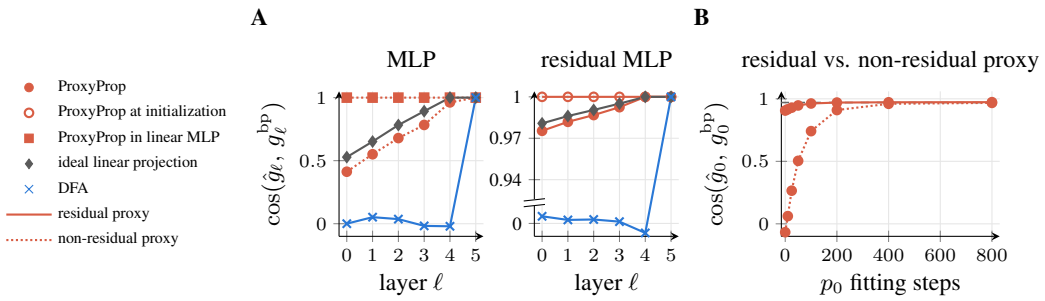

Figure 2: **Numerical illustration of the theory. 2A**: (Left) Alignment of the ProxyProp and backpropagation gradients in a random 6-layer MLP processing MNIST digits. (Right) Same as left, but the alignment is higher because of the skip connections. **2B**: In the same residual setting, gradient alignment improves when the proxy objective is optimized (the network remains frozen). The residual proxy (solid) aligns already at the first iteration and converges faster.

Proof. Since the network is linear, each layer activation z_ℓ is simply a linear transformation of the input and is therefore also Gaussian. The covariance at layer ℓ follows $\Sigma_\ell = W_\ell \Sigma_{\ell-1} W_\ell^T$ where W_ℓ is the linear feedforward weight. With full-row-rank W_ℓ, $\Sigma_{\ell-1}$ being positive definite implies positive definiteness of Σ_ℓ. As Σ_0 is positive definite, Σ_ℓ is positive definite for all ℓ. Moreover, each F_ℓ is linear and therefore also locally Lipschitz. By Proposition3.1, $P_\ell^* = \mathbb{E}[\frac{\partial z_L}{\partial z_\ell}]$. Because F_ℓ is a linear function, $\frac{\partial z_L}{\partial z_\ell}$ has no dependence on z_ℓ, so $\frac{\partial z_L}{\partial z_\ell} = \mathbb{E}[\frac{\partial z_L}{\partial z_\ell}] = P_\ell^*$. Substituting this into Equation 1, we find that the ProxyProp gradient is exactly the backpropagated gradient. □

Numerical verification in linear networks. We verify Proposition 3.1 and Corollary 3.2 in a toy setting with linear networks. We consider a six-layer MLP fed with batched images from the MNIST dataset (LeCun et al., 2010). When the MLP activation function is linear (squares in Figure 2A left), the gradient g_θ^{loc} and the true gradient g_θ^{BP} align perfectly as predicted by Corollary 3.2.

We then study the accuracy of a linear proxy in non-linear networks. In this case, even perfect estimation of the expected Jacobian $\mathbb{E}[\partial z_L/\partial z_\ell]$ does not mean good estimation of the gradient. With non-linearities, the expected Jacobian does not equal the sample-specific Jacobian $\partial z_L/\partial z_\ell$ (Delrocq et al., 2026). Indeed, the gradient alignment is imperfect in our toy setting (circles in Figure 2A), but ProxyProp approaches the best linear projection achievable (computed using the true gradient: $P' = \text{argmin}_P ||\partial\mathcal{L}/\partial z_\ell - P^\top \partial\mathcal{L}/\partial z_L||^2$). As a comparison, DFA (Nøkland, 2016) (which uses a random projection) has little alignment (crosses). In our toy MLP setting, we demonstrate that the proxy distillation loss of Equation 2 builds alignment without explicitly minimizing the difference of the gradients.

Gradient consistency in non-linear proxies. As shown in Figure 2A, even the very best linear proxy cannot yield perfect alignment in non-linear network. The problem is further exacerbated in transformers taking sequential inputs. Token-wise linear proxies are time invariant, so linear projections cannot track the temporal computation performed by attention layers in transformers. In such cases, the proxy ideally is a non-linear sequence model, so we choose p_ϕ^ℓ to be a transformer itself but with smaller depth and width. We use trainable linear projectors to map the hidden dimension of the network to that of the proxies.

Gradient consistency in residual proxies. Modern architectures (e.g. transformers) rely on residual connections (He et al., 2016). We design ProxyProp to leverage this to improve the gradient estimations. Below we show that the gradient approximation of ProxyProp is exact in residual networks at initialization (Remark 3.3), and the error is bounded whenever the residual contribution at each layer is small in comparison with the corresponding skip connection (Proposition 3.4).

Let's first consider the initialization. The preferred initialization in residual networks and transformers enforces $f_\theta^\ell(z_\ell) = z_\ell$ by initializing to zero the output weights (Zhang et al., 2019) or by using a scalar to gate the output activation (Goyal et al., 2017; Bachlechner et al., 2021). At least during the early phase of training, the skip connections dominate the computation $z_L \approx z_\ell$, which reflects during gradient propagation as $\partial\mathcal{L}/\partial z_\ell \approx \partial\mathcal{L}/\partial z_L$. In ProxyProp, a trivial identity proxy is therefore

sufficient to propagate an accurate approximation during the first training iterations. More generally, we can choose any (non-linear) residual proxy of the form $p_\phi^\ell(z_\ell) = z_\ell + q_\phi^\ell(z_\ell)$ with $q_\phi^\ell = 0$ at initialization. We start with a simple remark regarding the benefit of this strategy (this proof is a direct consequence of the large network and the proxy being initialized to identity by construction):

Remark 3.3. *Consider any (non-linear) residual network and a residual proxy using standard zeroed residual initialization. Then, ProxyProp computes the exact error gradient at the first iteration.*

More broadly in any residual architecture, we can bound the gradient approximation error of ProxyProp whenever the residual contributions are small in comparison with the skip connections. To formalize this, we assume that each layer is residual $f_\theta^\ell(z_\ell) = z_\ell + r_\theta^\ell(z_\ell)$, such that the deviations of the residual $||r_\theta^\ell(z) - r_\theta^\ell(z')||$ are always smaller than the deviations in the skip connections $||z - z'||$. When z and z' are close, this is equivalent to assuming that the operator norm of the Jacobian of the residual is bounded $||\partial r_\theta/\partial z||_{op} \leq \epsilon$, whereas the norm of the Jacobian through the skip connection is one. This is expected when the weights are still close to the zeroed initialization of Remark 3.3. Then, if we parameterize the proxy $p_\phi^\ell(z_\ell) = z_\ell + q_\phi^\ell(z_\ell)$ such that $||\partial q_\phi/\partial z||_{op} \leq \epsilon$, we have (detailed proof in the Appendix B):

Proposition 3.4. *Consider a deep (non-linear) residual network which is dominated by the skip connections (at all layers we assume $||\partial r_\theta/\partial z||_{op} \leq \epsilon$). Then, given a fixed depth L, if we choose a (non-linear) residual proxy with $||\partial q_\phi/\partial z||_{op} \leq \epsilon$, there exists a constant C (given explicitly in the Appendix B) such that the gradient estimation error is bounded as follows:*

$$||g_{\theta_\ell}^{BP} - g_{\theta_\ell}^{loc}|| \leq C\epsilon + O(\epsilon^2)$$

Numerical advantage of residual proxy. To verify the impact of the residual proxy during pretraining, we construct a 6-layer MLP with skip connections at each layer: $z_{\ell+1} = z_\ell + W_{\ell,2}\text{ReLU}(W_{\ell,1}z_\ell)$. We confirm Remark 3.3 by showing that with zeroed initialization ($W_{\ell,2} = 0$), the gradients are perfectly aligned at all layers in Figure 2A (right). Away from initialization (non-zero $W_{\ell,2}$), $||\partial r_\theta/\partial z||_{op}$ is not necessarily small, but gradient alignment of the residual proxy is above 0.9, approaching the best possible alignment. Even when the distillation loss is not yet fully minimized, the residual proxy gradient aligns remarkably well (Figure 2B). Distillation eventually improves the alignment further.

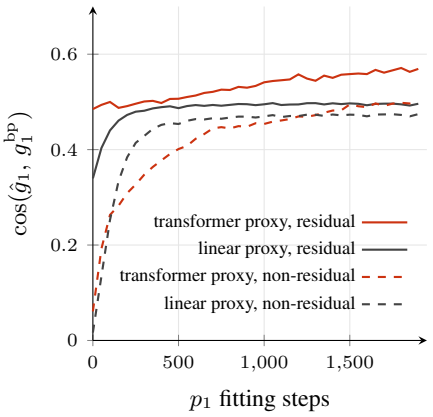

Figure 3: **Non-linear proxy with residual connection improves gradient alignment in trained transformers.** Cosine similarity between the ProxyProp and the backpropagation gradients for the second block of a 12-layer ViT (partitioned into three blocks of four layers).

With trained transformers, approaching perfect alignment is much harder, though the residual parameterization still shows a clear advantage. To show this empirically, we trained a ViT on CIFAR-100 classification (Krizhevsky, 2009) for 100 epochs with end-to-end backpropagation. Then, we partition the ViT into three blocks and train a proxy on the penultimate block with a frozen backbone. We observe in Figure 3 that, firstly, transformer proxies are better than linear proxies, and secondly, a residual proxy parameterization improves the alignment.

4 Accuracy/Memory Trade-off in Transformers

Studies in vision transformers. To test empirically if the theoretical consistency of the proxy gradient translates to training transformers, we first implement a 12-layer ViT (Dosovitskiy et al., 2021) with embedding dimension $d = 128$ on CIFAR-100. The network is partitioned into $L = 3$, 6, or 12 blocks, the three cases being denoted 3×4, 6×2, and 12×1 respectively to represent the number of layers per block. Gradients are computed either with ProxyProp or alternative local learning algorithms. ProxyProp parameterizes a proxy for each block as a small transformer of lower dimension $d = 64$. ProxyProp (3×4) achieves performance close to backpropagation while reducing the VRAM requirements by 62% (Figure 1B).

Table 1: **ProxyProp offers a better trade off between accuracy and memory cost**. We report here the results on CIFAR-100 with a 12-layer ViT. We use a relatively large batch of 512 to highlight the memory gain of the activation vector. Our backpropagation baseline with the same architecture and training hyperparameters achieved 66.13% accuracy with 12.30 GB of peak VRAM usage. The accuracy number is in bold when it is highest or non-significantly lower (one-sided paired t-test over 5 seeds, Holm-corrected; $p < 0.05$).

Method	top-1 accuracy			VRAM (GB)
	3×4	6×2	12×1	3×4/6×2/12×1
ProxyProp	**64.90**±0.44	**61.51**±0.19	**57.58**±0.40	4.64 / 2.76 / 1.90
with DNI style loss	**64.43**±0.54	60.42±0.63	**58.31**±0.60	13.28 / 7.92 / 5.10
residual linear proxy	63.18±0.56	58.86±0.32	47.67±0.27	4.62 / 2.69 / 1.72
identity proxy	58.15±0.84	57.08±0.57	46.07±0.41	4.62 / 2.69 / 1.72
Backprop (one block)	63.93±0.60	60.00±0.54	50.33±0.32	4.44 / 2.47 / 1.49
Local loss (Belilovsky et al., 2019)	64.06±0.26	60.15±0.45	51.96±0.44	4.57 / 2.63 / 1.66
DFA (Nøkland, 2016)	55.32±0.19	49.96±0.37	35.66±0.24	4.62 / 2.74 / 1.87
DRTP (Frenkel et al., 2021)	58.56±0.41	54.66±0.28	34.97±0.66	4.62 / 2.74 / 1.87
DNI (Jaderberg et al., 2017)	41.39±0.45	33.11±0.34	16.67±3.58	4.66 / 2.72 / 1.76

To justify the design choices of the proxies, we implement ProxyProp variants and compare their classification performance (Table 1). The simplest variant is when the proxy is reduced to identity (the error gradient $\partial\mathcal{L}/\partial z_L$ is applied at every layer without transformations). Despite its simplicity, it outperforms local learning algorithms that use random fixed matrices to train the blocks (see DFA and DRTP in Table 1). This observation supports the advantage of leveraging the residual structure when estimating local gradients (Remark 3.3 and Proposition 3.4). Consistent with our measurements on gradient alignment, non-linear transformer proxies perform better than linear ones.

Then, we examine if the simple proxy distillation objective could be replaced with loss functions that more explicitly target the gradient. We consider the DNI gradient bootstrapping loss from Jaderberg et al. (2017). Following their method, we train an auxiliary MLP to estimate a gradient bootstrapped recursively from the auxiliary network of the next layer: with our ViT architecture this method is not stable. Instead, we use ProxyProp but replace the distillation loss (Equation 2) with the DNI loss that explicitly estimates the global gradient. Despite the increase in memory from the second-order derivatives, it does not bring any significant improvement over our simpler distillation loss (see Table 1). This empirical finding supports, along with Proposition 3.1 and Corollary3.2, that proxy distillation is a simple and efficient way to align the local gradients with the global gradients.

We measured for all methods the VRAM occupied in the GPU for the various training algorithms. Given a relatively large batch size, the memory consumption is dominated by the storage of the network activations to illustrate the benefits of the algorithm. By choosing the number of layers for each block in the network, we can navigate the trade-off between network performance and memory consumption (either 3, 6, or 12 blocks). While ProxyProp pays a small storage to propagate gradients through the proxy, it sits on the Pareto front of the accuracy/memory trade-off (Figure 1B).

Fine-tuning in large language models. We identify that LLM fine-tuning is a relevant use case where ProxyProp can be decisive. While LLMs are pretrained in datacenters with extensive resources, fine-tuning these models is practicable with consumer-grade GPUs. Choice of model and training context length are often limited by the VRAM capacity. For fine-tuning, the memory storage can be roughly divided into three main categories: the weights, the optimizer, and the model activations. The memory footprint of the weights may be reduced via quantization (Dettmers et al., 2023) and the optimizer state via low-rank adapters (LoRA (Hu et al., 2022)); we study how local learning reduces the memory footprint of the activations.

A popular method to almost entirely eliminate the storage of the activations is to estimate gradients with repeated forward passes. These methods are often referred to as zeroth-order optimization techniques (e.g. MeZO, Malladi et al. (2023); Tan et al. (2025)); they are cheap in memory, but they require repeated evaluations of each sample with noisy parameter perturbations. We show below that ProxyProp offers a practical alternative.

First we demonstrate that ProxyProp is able to fine-tune LLMs. We choose popular datasets and models with existing reference baselines for MeZO and backpropagation, then we fine-tune language

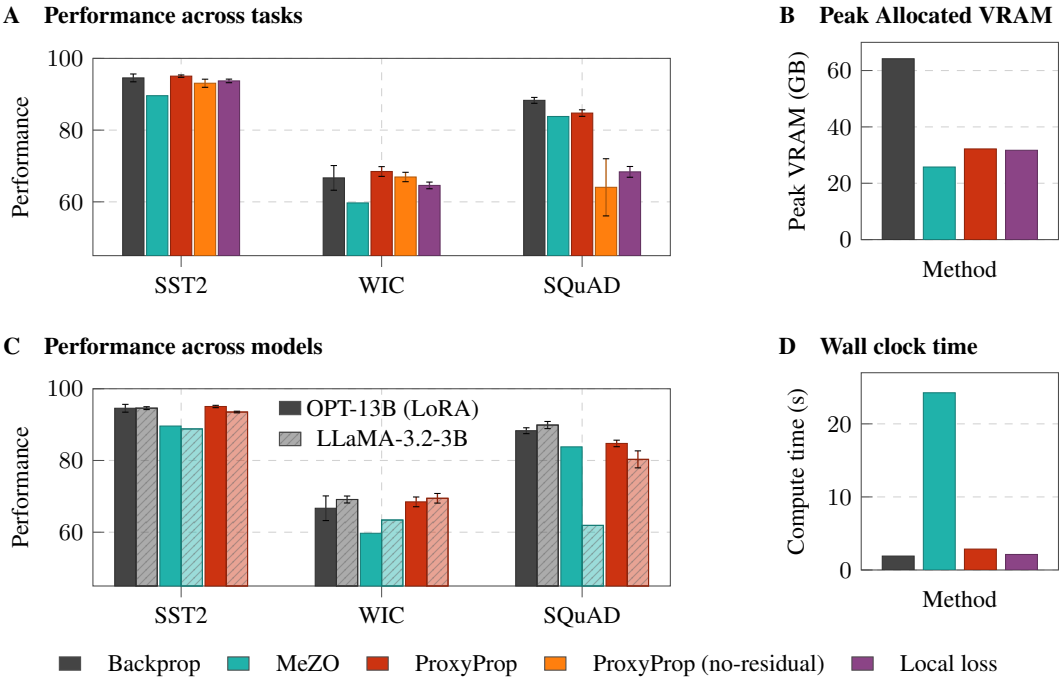

Figure 4: **ProxyProp fine-tunes LLMs with better tradeoff between performance, memory, and compute time. 4A** OPT-13B fine-tuned with rank-8 LoRA by Backprop, MeZO, and local algorithms; SST-2 and WiC (classification tasks) report accuracy, and SQuAD (question answering task) reports F1. MeZO values are taken from Malladi et al. (2023); Tan et al. (2025). Error bars are the 95% confidence interval computed from 5 different random seeds. **4C** Performance on OPT-13B with LoRA (solid) and LLaMA-3.2-3B (hatched). **4B, 4D** Peak VRAM and compute time for one training step of input batch with size 8, sequence length 800 on A100 GPU. MeZO's compute time is scaled up by its 16x number of training steps used in Malladi et al. (2023).

models OPT-13B (Zhang et al., 2022) and LLaMA-3.2-3B (Dubey et al., 2024) on three datasets (1000 examples each): SST2, WIC, and SQuAD (Socher et al., 2013; Wang et al., 2019; Rajpurkar et al., 2016). We split the entire model into either 7 or 8 blocks and use proxies of dimension 64 or 80 depending on the model (details in Appendix C). ProxyProp approaches the performance of backpropagation across the 3 tasks and 2 architectures, and also achieves higher performance than the results MeZO reports (Figure 4A, 4C). We also confirm that alternative local algorithms struggle to achieve a comparable performance on these fine-tuning benchmarks (we compare against ProxyProp without the residual parameterization and against block-wise local losses).

Our experiments show that ProxyProp uses resources efficiently. The backpropagation baseline is implemented with the standard HuggingFace Trainer. Towards a fairer comparison, we inherit the trainer, swapping out PyTorch backpropagation with ProxyProp's computation of the gradients. For OPT-13B, we use LoRA to reduce the memory consumption of the optimizer. On the same hardware, ProxyProp is roughly as fast as backpropagation (approximately $\times 1.5$ slower). In comparison, zeroth-order optimization has a compute time that scales with the requirement of more training steps. Even 16 times as many steps (as reported in Malladi et al. (2023)), which means eight times the wall-clock time, are not sufficient for MeZO to reach the performance of ProxyProp. In terms of memory, MeZO requires less VRAM because it only needs to store the network weights (approximately 25.7 GB). ProxyProp needs 32 GB and backpropagation 64 GB (see Figure 4B). ProxyProp achieves a strong memory-efficient performance without compromising training time.

To demonstrate the compatibility of ProxyProp with other commonly used memory-efficient techniques, we combine it with LoRA, chunked loss, and gradient checkpointing (the default Hugging Face implementation that caches only the output of each transformer layer). We compare the memory scaling of training LLaMA-3.1-8B by backpropagation and by ProxyProp with artificial data at

varying sequence lengths. As shown in Figure 5, ProxyProp with 4 blocks could fit in sequences over 50% longer than backpropagation. As the number of blocks increases, ProxyProp fits in longer sequence and approaches the memory scaling of inference.

5 Discussion

We analyzed theoretically how to compute local and memory-efficient gradient estimates in transformers. We present ProxyProp as a local learning algorithm which estimates the global gradient with theoretical guarantees. In practice, this algorithm trains ViTs and fine-tunes LLMs to near-backpropagation accuracy on a low memory budget. Below we speculate the potential impact of local learning beyond what we have presented in previous sections.

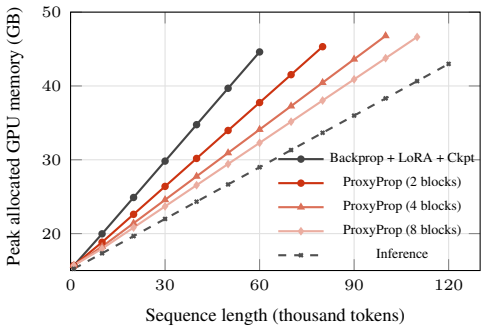

Figure 5: **ProxyProp approaches inference memory usage.** We measure peak memory allocation when fine-tuning a LLaMA-8B model with varying-length sequence input. All baselines benefit from the standard memory-efficient fine-tuning implementation of the HuggingFace transformer library, including rank-8 LoRA, gradient checkpointing, and cross-entropy loss chunking. For ProxyProp we use a residual proxy transformer with 64 dimensions.

Local learning for very deep, looped, or diffusion architectures. The memory benefits of local learning are already visible on moderately deep LLM architectures (dozens of layers), but ProxyProp memory consumption is constant with the number of layers. From the viewpoint of memory, the benefits increase with deeper networks. Local learning has the potential to scale to much deeper architectures which are hard to train with standard backpropagation.

A case of interest is a looped transformer in which repeating the recurrence is effectively increasing the depth of the network. To avoid increasing the memory storage, the gradients are not propagated through all of the loops; rather, the gradient is propagated only through the last (or last few) loop iterations Yang et al. (2024). Local proxy propagation learning would be adapted to this setting, as it offers theoretical insights and a method to compute gradients which are aware of all the repeated loops without increasing memory storage. From another perspective, a single block in residual models is analogous to a diffusion step. Pushing this analogy, Li et al. (2026) and Shing et al. (2026) train deep residual architectures using diffusion training schemes. Reciprocally, our analyses might inspire methods to compute global gradients from a single local diffusion step.

Decoupling parameter updates. The potential of local learning is not limited to memory saving. By solving the backward locking problem, local learning also enables model-parallelism during training (Jaderberg et al., 2017; Belilovsky et al., 2020). While the computation speed on a single device is roughly similar between ProxyProp and backpropagation (see Table A1), decoupling becomes relevant when training across devices. As described, ProxyProp from Algorithm 1 preserves a weak coupling. We see at least two directions to solve this issue: firstly, the pre-forward pass could be computed ahead of time for a large batch without coupling; with mini-batches then being dispatched in order and iterated. Hence, the algorithm can be applied if one accepts a lag between the loss gradients and the local block that it is then sent to. Alternatively, we describe in Algorithm 2 the fully unlocked ProxyProp variant that uses a pretrained proxy to train the network. Although we have not studied distributed implementations of our algorithm at the moment, we feel that it is a promising direction for future work.

AI USE STATEMENT

We used generative AI tools to help review the literature exhaustively. We have used AI to brainstorm mathematical justifications for the algorithm we had planned to implement. We used AI to generate code, but we implemented ourselves the critical core functions of the local learning algorithm which is central to this publication. Less critical code, such as the implementation of a toy example or standard baselines, was quality controlled by the entire team using numerical and visual verifications. We wrote the first versions of every mathematical proof ourselves, and used AI to improve the writing or mathematical consistency and verify our claims. All the final proofs have been reviewed by all the authors.

We have not used LLMs to seed or design the research direction. We have not used AI to write text or ideas. All the text was written manually without AI, but AI was used to find typographic mistakes. All the AI corrections were reviewed each time by one of the authors. We have not used AI to generate autonomously the experiments or optimize the performance autonomously. We did not use AI to recommend specific data analysis strategies, numerical experiments or visualization. Provided with data from our experiments, we used AI to help format figures in tikz. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Upon acceptance of the paper, we will publish the PyTorch implementation and local learning wrapper for the Trainer class of the Hugging Face library on GitHub. The datasets used in all experiments are freely and publicly available. All authors have been able to use our algorithm on different datasets and successfully trained a neural network model with consistent training speed and memory RAM requirement. The results that are claimed were consistent across datasets, even when the experiments were implemented by different authors.

ACKNOWLEDGMENTS

REFERENCES

Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. *Advances in neural information processing systems*, 32, 2019.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in artificial intelligence*, pp. 1352–1361. PMLR, 2021.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pp. 583–593. PMLR, 2019.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pp. 736–745. PMLR, 2020.

Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

Colin Bredenberg, Ezekiel Williams, Cristina Savin, Blake Richards, and Guillaume Lajoie. Formalizing locality for normative synaptic plasticity models. *Advances in neural information processing systems*, 36:5653–5684, 2023.

Xing Chen, Dongshu Liu, Jérémie Laydevant, and Julie Grollier. Self-contrastive forward-forward algorithm. *Nature Communications*, 16(1):5978, 2025.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *Advances in neural information processing systems*, 30, 2017.

Ariane Delrocq, Wu S Zihan, Guillaume Bellec, and Wulfram Gerstner. Self-supervised local learning rules learn the hidden hierarchical structure of high-dimensional data. *arXiv preprint arXiv:2605.18557*, 2026.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Matthew Filipovich, Alessandro Cappelli, Daniel Hesslow, and Julien Launay. Scaling laws beyond backpropagation. In *I Can't Believe It's Not Better Workshop: Understanding Deep Learning Through Empirical Falsification*, 2022. URL https://openreview.net/forum?id=8eYNb2iUqb7.

Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in Neuroscience*, Volume 15 - 2021, 2021. ISSN 1662-453X. doi: 10.3389/fnins.2021.629892. URL https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2021.629892.

Liron Gelbard, Shay Landis, David Yannai, and Assaf Touboul. Unlocking the power of layer-by-layer training and fine- tuning. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=p5ObETPuTi.

Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 44(3):33–39, 2024.

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. *arXiv preprint arXiv:1706.02677*, 2017.

Manu Srinath Halvagal and Friedemann Zenke. The combination of hebbian and predictive plasticity learns invariant object representations in deep sensory networks. *Nature Neuroscience*, pp. 1–10, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Donald O Hebb. The organization of behavior. *New York*, 1949.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2(3):5, 2022.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Bernd Illing, Jean Ventura, Guillaume Bellec, and Wulfram Gerstner. Local plasticity rules can learn deep representations using self-supervised contrastive predictions. *Advances in Neural Information Processing Systems*, 34:30365–30379, 2021.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pp. 1627–1635. PMLR, 2017.

Oscar Key, Jean Kaddour, and Pasquale Minervini. Local lora: Memory-efficient fine-tuning of large language models. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*, 2023.

Daniel J Korchinski, Alessandro Favero, and Matthieu Wyart. Learn from your own latents and not from tokens: A sample-complexity theory. *arXiv preprint arXiv:2605.27734*, 2026.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Qinyu Li, Yee Whye Teh, and Razvan Pascanu. Noprop: Training neural networks without back-propagation or forward-propagation. In *Conference on Lifelong Learning Agents*, pp. 525–544. PMLR, 2026.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1): 13276, 2016.

Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Back-propagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. *Advances in neural information processing systems*, 32, 2019.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. A history of spike-timing-dependent plasticity. *Frontiers in synaptic neuroscience*, 3:4, 2011.

Kevin Max, Laura Kriener, Garibaldi Pineda García, Thomas Nowotny, Ismael Jaras, Walter Senn, and Mihai A Petrovici. Learning efficient backprojections across cortical hierarchies in real time. *Nature Machine Intelligence*, pp. 1–12, 2024.

Ali Momeni, Babak Rahmani, Matthieu Malléjac, Philipp Del Hougne, and Romain Fleury. Backpropagation-free training of deep physical neural networks. *Science*, 382(6676):1297–1303, 2023.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *Advances in Neural Information Processing Systems*, 37:41076–41102, 2024.

James M Murray. Local online learning in recurrent networks with random feedback. *Elife*, 8: e43299, 2019.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In *International conference on machine learning*, pp. 4839–4850. PMLR, 2019.

Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16:795876, 2022.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Makoto Shing, Masanori Koyama, and Takuya Akiba. Diffusionblocks: Block-wise neural network training via diffusion interpretation. In *International Conference on Learning Representations*, volume 2026, pp. 95053–95074, 2026.

Shoaib Siddiqui, David Krueger, Yann LeCun, and Stephane Deny. Blockwise self-supervised learning at scale. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=M2m618iIPk.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Suraj Srinivas and François Fleuret. Knowledge transfer with jacobian matching. In *International conference on machine learning*, pp. 4723–4731. PMLR, 2018.

Zhe Su and Giacomo Indiveri. Elfcore: A 28nm neural processor enabling dynamic structured sparse training and online self-supervised learning with activity-dependent weight update. In *2025 IEEE European Solid-State Electronics Research Conference (ESSERC)*, pp. 13–16. IEEE, 2025.

Qitao Tan, Jun Liu, Zheng Zhan, Caiwen Ding, Yanzhi Wang, Xiaolong Ma, Jaewoo Lee, Jin Lu, and Geng Yuan. Harmony in divergence: Towards fast, accurate, and memory-efficient zeroth-order LLM fine-tuning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Rx6m16By6l.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Ziao Wang, Kilian Müller, Matthew Filipovich, Julien Launay, Ruben Ohana, Gustave Pariente, Safa Mokaadi, Charles Brossollet, Fabien Moreau, Alessandro Cappelli, et al. Streamlined optical training of large-scale modern deep learning architectures with direct feedback alignment. *Proceedings of the National Academy of Sciences*, 123(20):e2532022123, 2026.

James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5): 1229–1262, 2017.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International conference on learning representations*, volume 2024, pp. 42195–42214, 2024.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gsz30cKX.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

Wu S. Zihan, Ariane Delrocq, Wulfram Gerstner, and Guillaume Bellec. Can local learning match self-supervised backpropagation? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0QsbcZKCQ0.

A SCALING OF MEMORY AND COMPUTE

Table A1: Scaling of compute and memory cost with training an L-block network. * For transformer, the optimal activation checkpointing is not $\mathcal{O}(\sqrt{L})$ because attention and MLP within each layer also requires plenty of memory storage. A common practice is to store the output of each transformer layer and recompute the activations of attention and MLP, which make the activation storage $\mathcal{O}(L)$.

Method	Forward FLOPs	Backward FLOPs	Activation storage
Backprop (BP)	$\mathcal{O}(L)$	$\mathcal{O}(L)$	$\mathcal{O}(L)$
BP + checkpointing	$\mathcal{O}(2L)$	$\mathcal{O}(L)$	$\mathcal{O}(\sqrt{L})^*$
ProxyProp	$\mathcal{O}(2L)$	$\mathcal{O}(L)$	$\mathcal{O}(1)$

Algorithm 2: Description of the fully unlocked ProxyProp algorithm. The input z_0 is given. $p_\phi^L(z_l)$ is always the identity.

Pretrain the proxy parameters ϕ with the distillation loss from Equation 2
for $\ell = 1$ **to** L **do**
  $z_\ell \leftarrow f_\ell(z_{\ell-1})$;
  Compute the proxy $p_\phi^\ell(z_\ell)$;
  Compute loss $\mathcal{L}(\theta_h, p_\phi^\ell(z_\ell), y)$;
  Update θ_ℓ with local gradient using $\partial\mathcal{L}/\partial\theta_\ell$;
  if $\ell = L$ **then**
    Update θ_h;
  delete $z_{\ell-1}$;

B PROOF FOR PROPOSITION 3.4

We use the notation of the main text: the network is partitioned into blocks $z_\ell = f_\theta^\ell(z_{\ell-1})$, each block is residual $f_\theta^\ell(z) = z + r_\theta^\ell(z)$, and the proxy is residual $p_\phi^\ell(z) = z + q_\phi^\ell(z)$. We assume that every residual above ℓ, that is $r_\theta^{\ell'}$ for $\ell' > \ell$, and the proxy residual q_ϕ^ℓ (linear or not) are all ϵ-Lipschitz, i.e. $\|r_\theta^{\ell'}(z) - r_\theta^{\ell'}(z')\| \le \epsilon\|z - z'\|$ and $\|q_\phi^\ell(z) - q_\phi^\ell(z')\| \le \epsilon\|z - z'\|$ for all z, z'. We write $R^{\ell'} = \partial r_\theta^{\ell'}/\partial z$ and $Q^\ell = \partial q_\phi^\ell/\partial z$ for the Jacobians of these residual maps, so that $\|R^{\ell'}\|_{\mathrm{op}} \le \epsilon$ and $\|Q^\ell\|_{\mathrm{op}} \le \epsilon$ everywhere ($\|\cdot\|_{\mathrm{op}}$ denotes the operator norm). By definition of the backpropagated gradient and of ProxyProp's gradient estimate in Equation 1, we recall that:

$$g_{\theta_\ell}^{BP} = \frac{\partial f_\theta^\ell}{\partial \theta}^\top \frac{\partial z_L}{\partial z_\ell}^\top \frac{\partial \mathcal{L}}{\partial z_L}, \qquad g_{\theta_\ell}^{loc} = \frac{\partial f_\theta^\ell}{\partial \theta}^\top \frac{\partial p_\phi^\ell}{\partial z_\ell}^\top \frac{\partial \mathcal{L}}{\partial z_L}. \tag{5}$$

Expanding the stacked Jacobian. By the chain rule, $\frac{\partial z_L}{\partial z_\ell} = \prod_{\ell'=\ell+1}^{L}(I + R^{\ell'})$, ordered with decreasing index from left to right, a product of $L - \ell$ near-identity factors, one for each block above ℓ. Expanding this product as a sum over which factors contribute $R^{\ell'}$ and which contribute I gives $2^{L-\ell}$ paths inside the computational graph (at each layer a path either goes through a skip connection or through a residual map). The leading path π_0, which goes only through identities, contributes exactly I, and is what the naive identity proxy ($p_\phi^\ell(z_\ell) = z_\ell$) estimates. Grouping the remaining paths by their number of non-identity factors:

$$\frac{\partial z_L}{\partial z_\ell} = I + \sum_{\ell'=\ell+1}^{L} R^{\ell'} + \sum_{\ell<\ell'<\ell''\le L} R^{\ell''} R^{\ell'} + \dots. \tag{6}$$

Since $\|R^{\ell'}\|_{\mathrm{op}} \le \epsilon$ for all ℓ', the sum of the $\binom{L-\ell}{k}$ terms with $k \ge 2$ paths is bounded, by submultiplicativity of the operator norm, by $\sum_{k\ge 2}\binom{L-\ell}{k}\epsilon^k = O(\epsilon^2)$, so:

$$\frac{\partial z_L}{\partial z_\ell} = I + \sum_{\ell'=\ell+1}^{L} R^{\ell'} + O(\epsilon^2)\,. \tag{7}$$

Likewise, by definition of the residual proxy, $\frac{\partial p_\phi^\ell}{\partial z_\ell} = I + Q^\ell$ exactly, with $\|Q^\ell\|_{\mathrm{op}} \le \epsilon$.

Comparing ProxyProp to backpropagation. Subtracting the two expansions:

$$\frac{\partial z_L}{\partial z_\ell} - \frac{\partial p_\phi^\ell}{\partial z_\ell} = \sum_{\ell'=\ell+1}^{L} R^{\ell'} \;-\; Q^\ell \;+\; O(\epsilon^2)\,, \tag{8}$$

so that, using the triangle inequality and $\|R^{\ell'}\|_{\mathrm{op}}, \|Q^\ell\|_{\mathrm{op}} \le \epsilon$:

$$\Big\| \frac{\partial z_L}{\partial z_\ell} - \frac{\partial p_\phi^\ell}{\partial z_\ell} \Big\|_{\mathrm{op}} \le (L-\ell)\,\epsilon + \epsilon + O(\epsilon^2) = (L-\ell+1)\,\epsilon + O(\epsilon^2)\,. \tag{9}$$

Finally, using submultiplicativity of the operator norm, and setting $C = (L-\ell+1)\,\|\partial f_\theta^\ell/\partial\theta\|_{\mathrm{op}}\,\|\partial\mathcal{L}/\partial z_L\|$:

$$\|g_{\theta_\ell}^{BP} - g_{\theta_\ell}^{loc}\| \le \Big\| \frac{\partial f_\theta^\ell}{\partial \theta} \Big\|_{\mathrm{op}} \Big\| \frac{\partial z_L}{\partial z_\ell} - \frac{\partial p_\phi^\ell}{\partial z_\ell} \Big\|_{\mathrm{op}} \Big\| \frac{\partial \mathcal{L}}{\partial z_L} \Big\| \le C\,\epsilon + O(\epsilon^2)\,, \tag{10}$$

which is the claimed bound. The argument never used linearity of q_ϕ^ℓ: only $\|Q^\ell\|_{\mathrm{op}} \le \epsilon$ was needed, so it applies equally to non-linear proxies. The naive identity proxy ($q_\phi^\ell = 0$, so $Q^\ell = 0$) is the valid special case.

C Experiment Details

C.1 Numerical Verification

A 6-layer MLP on MNIST with width 64 and batch size 4096. Each layer is a transformer-style branch $W_{l,2}\mathrm{ReLU}(W_{l,1}, z_{l-1})$ with hidden width 256 either with or without the skip connection.

Figure 2A solves a linear proxy closed-form to z_L either on the plain network or to $z_L - z_l$ on the residual network. The alignment is measured between the local gradient at each layer through the proxy and the true gradient.

Figure 2B freezes the residual network at random and measures the alignment of layer 0 when the linear proxy is trained by Adam for 800 steps. The flat line denotes the closed-form solve.

C.2 ViT

C.2.1 Specifications

All CIFAR-100 experiments use a 12-layer vision transformer of width 128, 4 attention heads, patch size 4 with classification on a final mean token readout. For local methods, the final LayerNorm is placed in the classifier rather than in the last block so that local methods read raw activations at each partition boundary. The classifier is zero-initialized, as in the original ViT.

Training uses the AdamW optimizer with batch size 128 at learning rate 5×10^{-4} with 10% linear warm-up and cosine decay over 100 epochs. ProxyProp uses a single transformer at width 64 and is trained by the same optimizer as the backbone. We report last-epoch accuracies.

To highlight the savings in memory in Figure 1B, we increase the batch size to 512 and measure peak memory usage for one epoch.

C.3 FINETUNING

We follow the training setup in MeZO (Malladi et al., 2023) and build our experiments based on their published repository. All models are finetuned for 10 epochs with batch size 8, and the epoch with the best validation loss is used for evaluation on the test dataset. For OPT-13B, we use rank-8 LoRA and split the 40-layer network into 8 blocks. We use the Adam optimizer and swept the learning rate $(3 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4})$ to select the optimal learning rate. The proxies are parameterized as: a linear projection down to 80 dimension, a small OPT transformer layer with 80 hidden dimension and 40 heads, and a linear projection back to network hidden dimension. The second linear projector is zero-initialized. A residual connection to each proxy is added unless specified. For LLaMA-3B, we split the 28 layer network into 7 blocks. The proxies are parameterized as a small LLaMA transformer layer with 64 hidden dimensions and linear projectors mapping to network hidden dimension. We use the Adam optimizer and swept the learning rate $(5 \times 10^{-6}, 10^{-5}, 3 \times 10^{-5})$ to select the optimal learning rate. For all experiments, parameters of the proxies are trained with the same learning rate as the network parameters. For the local loss objective, we use the shared classification head for all blocks, because separate classification heads would result in a large memory cost. We use bfloat16 mixed precision training. All other settings remain exactly the same as in Malladi et al. (2023). Experiments are performed on one 80G Nvidia H100 for SQuAD and one A100 for SST2 and WIC.

When profiling the memory, we train the network with synthetic random data. To reduce the large memory requirement of language head, we compute the cross-entropy loss by chunking the final activation along the sequence dimension into pieces of a length 512.

D ADDITIONAL ABLATIONS

D.1 VIT EXPERIMENTS

We perform further ablations in Table A2. At convergence, we find no significance between residual and non-residual parameterization of the proxy in the classification, despite the clear advantage of residual in fine-tuning tasks. Further, we find that by prepending some transformer layers to the head (Gelbard et al., 2026) and training the proxies to distill the input into the deep head is advantageous. In this case, z_L would not be the last layer of the network but the last k-th layer with the parameters of the last k layers as part of θ_h. We found that it could further improve performance at roughly the same memory cost.

To make ProxyProp "update unlocked," we use Minitron (Muralidharan et al., 2024) to prune the transformer layer above each block into a 64-wide transformer proxy at the beginning of each epoch. We train this proxy for 20% of the batch, with the backbone frozen. Then, we freeze the proxy, and train the backbone with the proxy frozen for the entire batch.

Finally, we offer another baseline where we train a single block of the partitioned network with the proxy attached with end-to-end by backpropagation on the classification loss under the same optimizer. In ProxyProp, the proxy represents the distilled downstream network; this baseline suggests this distillation is more useful than simply augmenting the block-wise local loss with an additional proxy.

D.2 FINETUNING

We also study the importance of proxy training in SQuAD fine-tuning tasks. We found that the identity proxy has already reached a good performance, but a trained proxy further improves the performance. Consistent with findings on ViT, prepending the last block with the loss head improves the performance.

Table A2: **Extended ablations and baselines of ViT on CIFAR-100.**

backprop		$66.13_{\pm 0.55}$		12.30
Method		top-1 accuracy		RAM (GiB)
	3×4	6×2	12×1	$3\times4/6\times2/12\times1$
ProxyProp	$64.90_{\pm 0.44}$	$61.51_{\pm 0.19}$	$57.58_{\pm 0.40}$	4.64 / 2.76 / 1.90
non-residual proxy	$64.86_{\pm 0.43}$	$61.46_{\pm 0.20}$	$57.73_{\pm 0.58}$	4.62 / 2.74 / 1.88
deep head	$65.47_{\pm 0.77}$	$63.38_{\pm 0.47}$	—	4.85 / 2.92 / —
fully-unlocked	$64.78_{\pm 0.42}$	$59.79_{\pm 0.20}$	$42.02_{\pm 1.28}$	4.60 / 2.67 / 1.71
closed-form solve (EMA)	$62.93_{\pm 0.59}$	$58.95_{\pm 0.47}$	$48.16_{\pm 0.43}$	4.62 / 2.69 / 1.72
closed-form solve	$58.56_{\pm 1.42}$	$46.40_{\pm 7.81}$	$31.92_{\pm 0.84}$	4.60 / 2.67 / 1.70
random residual linear proxy	$57.37_{\pm 0.63}$	$55.90_{\pm 0.39}$	$46.03_{\pm 0.25}$	4.62 / 2.69 / 1.72
random linear proxy	$21.03_{\pm 9.49}$	$13.02_{\pm 1.19}$	$13.87_{\pm 1.71}$	4.60 / 2.67 / 1.70
backprop (one block)	$63.93_{\pm 0.60}$	$60.00_{\pm 0.54}$	$50.33_{\pm 0.32}$	4.44 / 2.47 / 1.49
BP (one block + proxy)	$63.71_{\pm 0.46}$	$60.31_{\pm 0.26}$	$54.39_{\pm 0.44}$	4.44 / 2.47 / 1.49

Table A3: Performance of applying ProxyProp variants to finetuning OPT-13B (with LoRA-8) on SQuAD. All ProxyProp variants split the network into 8 blocks and use proxies with hidden dimension 80. The value after $\pm$ is the 95% confidence interval computed from 5 different random seeds.

Method	SQuAD Performance
Backprop	88.30 ± 0.82
ProxyProp	84.75 ± 0.91
ProxyProp with fixed identity proxy	82.15 ± 1.82
ProxyProp + last block in the head	86.20 ± 1.15