# PERTURBATION PROPAGATION: SCALABLE LOCAL LEARNING WITHOUT WEIGHT TRANSPORT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The growing energy demands of artificial neural networks on digital hardware increasingly limit the level of intelligence they can reach, motivating the search for potentially far more energy-efficient analog implementations. The brain offers an existence proof that a physical system can learn complex tasks at very low energy cost, but a biologically plausible learning algorithm that trains at scale remains elusive. Backpropagation is not biologically plausible: it relies on exact derivatives from mathematical models of neuron responses and on weight transport, which copies forward weights into the backward pathway. More biologically plausible alternatives struggle to match backpropagation's scalability and algorithmic simplicity. We propose Perturbation Propagation (PerProp), which keeps the forward-backward structure of backpropagation but replaces both operations with local measurements: neuron derivatives with slopes measured by small forward perturbations, and weight transposes with feedback matrices learned from random probes. Because each neuron acts on its own input, one pair of perturbed evaluations measures a whole layer's slopes; because weights are shared across examples, feedback matrices are learned once per update. In Transformer pretraining, PerProp scales at a rate comparable to backpropagation with model size and compute, and at a faster rate with data, batch size, and depth. On GPT and Mamba-2 language models, it outperforms the feedback-alignment and perturbation-based alternatives we evaluate, both in and out of distribution. On spiking, comparator, and probabilistic-bit networks, it rivals or exceeds surrogate-gradient training without hand-designed surrogate derivatives. On analog hardware, PerProp trains physical neural networks directly, learning entirely on the BrainScaleS-2 neuromorphic chip and substantially outperforming reward-modulated STDP. Component-level estimates for analog crossbars further project orders-of-magnitude lower training energy than GPU backpropagation, an advantage that grows with network size. PerProp thus offers a unified approach to scalable, local learning in digital and physical neural networks.

## 1 INTRODUCTION

Artificial intelligence now shapes daily life and drives frontier science, from mathematics and physics to chemistry, biology and materials(Davies et al., 2021; Jumper et al., 2021; Merchant et al., 2023). Much of the intelligence in an AI system has come from the large scale artificial neural networks (ANNs) with trillion parameters and a huge amount of training data, which together consume much energy, creating a concern of not only the high cost but also sustainability. Moreover, as models grow, the energy required to train and run them will ultimately limit the level of intelligence we can achieve using the current implementation of ANNs (since shrinking transistors no longer brings proportional reductions in operating voltage and energy per operation (Momeni et al., 2025)).

Would it be possible to have an alternative implementation of the ANNs that is much more energy efficient? The brain offers an existence proof that complex learning is possible at very low energy cost (Levy & Calvert, 2021), motivating the search for biologically plausible learning algorithms that can train at scale; if we could design such an algorithm, it would enable training, at scale, an alternative, potentially much more energy efficient, physical neural network (PNN) directly implemented on analog hardware.

However, Backpropagation, the conventional learning method, is not biologically plausible because it relies on exact derivatives from mathematical models of neuron responses and on weight transport, which copies forward connection strengths into the backward pathway; neither operation has an



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.

established biological mechanism. As a result, backpropagation cannot execute natively on physical neural networks, whose neuron responses are not described by exact mathematical models, and prior alternatives have not matched its scaling behavior.

The existing work addressing this challenge has only made limited progress. Existing approaches follow four broad strategies. Physics-aware training combines physical forward measurements with derivatives from a digital model (Wright et al., 2022). Equilibrium propagation and coupled learning derive updates from differences between settled states of the network (Scellier & Bengio, 2017; Stern et al., 2021). Zeroth-order methods estimate update directions from changes in the loss caused by perturbations (Malladi et al., 2023). Other approaches modify credit assignment: feedback alignment uses fixed random feedback connections, while forward–forward trains each layer with its own objective (Lillicrap et al., 2016; Hinton, 2022). However, a common deficiency of all these approaches is that, while biologically more plausible, none of them is as scalable as backpropagation, nor can they deliver the level of prediction accuracy as backpropagation.

In this paper, we propose a novel training algorithm, called Perturbation Propagation (PerProp), which is not only biologically plausible but also as scalable as Backpropagation with a similar algorithmic simplicity. PerProp uses forward responses to local perturbations to replace both neuron derivatives (with measured local slopes) and weight transposes (with learned feedback matrices). To find how sensitive each neuron is, it nudges the neuron's input slightly and records how much the output changes, much as one judges the steepness of a hill by taking a small step. To carry the error backwards, it sends small random signals forward through each layer and learns feedback connections from how the layer responds.

PerProp builds the backward path for the error entirely from measurements taken as the network runs forward. For the neurons, it nudges their inputs twice and compares the outputs, which gives the slope of every neuron in a layer at the same time. For the connections between layers, it sends small random signals forward and learns feedback connections by matching each signal with the response it causes. The measured slopes and learned feedback connections then carry the output error backward, so that each connection receives its own local adjustment. The feedback connections are shared across all training examples, so the cost of learning them is spread over the whole batch. Each adjustment combines three signals available at the connection: the activity arriving at it, the measured sensitivity of the neuron it feeds, and that neuron's error. This is the same three-factor form as learning rules proposed for biological synapses, such as reward-modulated STDP (Wunderlich et al., 2019). All of these learning steps can run on the BrainScaleS-2 neuromorphic chip itself: its analog neurons provide the measured responses, and an on-chip processor learns the feedback, passes the errors backward, and updates the connection strengths.

We systematically evaluate PerProp across three regimes: digital pretraining at Transformer scale, non-differentiable and spiking networks, and analog neuromorphic hardware. In Transformer pretraining, PerProp achieves scaling rates comparable to backpropagation as model size and training compute increase, while PerProp exhibits steeper scaling slopes than BP along the data, batch size, and depth axes, closing the loss difference as training scale grows. Among the alternatives to BP in our comparison, PerProp ranks first on both in-domain and out-of-distribution validation loss on GPT and Mamba-2, while maintaining competitive training speeds. It also broadens the training options for non-differentiable networks, including spiking neural networks, and achieves competitive accuracy against backpropagation with surrogate gradient. PerProp also learns entirely on the analog BrainScaleS-2 neuromorphic chip, reaching 62.1% accuracy on Yin–Yang versus 36.9% for reward-modulated STDP. Training in this manner also enables direct training of physical neural networks (PNNs), an emerging computing technology that, like the brain, computes through analog physical dynamics and promises orders of magnitude lower energy consumption than conventional digital hardware. Hardware-in-the-loop simulations demonstrate projected energy-efficiency gains reach up to $1{,}700\times$ for training and $70\times$ for inference relative to V100 GPU backpropagation training and an H100 inference reference, respectively. Consequently, PerProp offers a unified approach to scalable learning across digital and physical neural networks.

Overall, these results suggest that the proposed PerProp is a highly promising alternative to backpropagation and makes a small step towards the ambitious but necessary goal of building much more energy-efficient AI systems in the future, which is desirable for not only reducing the cost, but also making AI more sustainable and reaching a higher level of artificial intelligence.

# 2 PERTURBATION PROPAGATION

## 2.1 NETWORK SETUP

We consider an $L$-layer network with forward computation

$$z_\ell = W_\ell a_{\ell-1}, \qquad a_\ell = \phi_\ell(z_\ell), \qquad \ell = 1, \dots, L. \tag{1}$$

Here, $a_0$ is the input, $W_\ell$ contains the trainable weights of layer $\ell$ (biases are included by appending a constant input), and $\phi_\ell$ applies a nonlinearity to each neuron separately, $[\phi_\ell(z)]_j = \phi_{\ell,j}(z_j)$. The neuron responses need not be differentiable; for example, they may be step functions or stochastic binary units. Let $\mathcal{L}(a_L)$ denote a loss function of the network output, which training aims to minimize. This alternation of linear maps and per-neuron nonlinearities is the building block of standard feed-forward architectures, including the feed-forward and projection layers of Transformers; Section 2.3 extends the formulation to spiking neurons, whose responses depend on time.

## 2.2 PERTURBATION PROPAGATION FOR STANDARD NEURAL NETWORKS

Figure 1 illustrates the measurements, backward error propagation, and weight update.

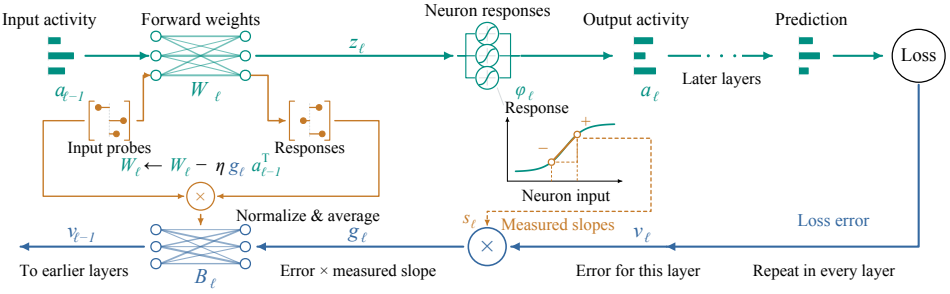


Figure 1: Activity flows forward through the weights and neurons; loss-derived errors return through measured neuron slopes and learned feedback connections. The local error $g_\ell$ and presynaptic activity $a_{\ell-1}$ determine the weight update, with learning rate $\eta$.

PerProp keeps the forward-backward structure of backpropagation. A forward pass through Eq. (1) produces the activations and the loss; a backward pass sends a per-neuron error signal from the output layer down to the first, and each weight is updated from the presynaptic activity it receives and the postsynaptic error it produces (Figure 1). PerProp differs from BP only in how the backward pass obtains its two ingredients: instead of computing neuron derivatives from a model of $\phi_\ell$ and using $W_\ell^\top$ as the backward weights, it *measures* both directly on the running network. Neuron slopes are measured once per forward pass by nudging each layer's input by $\pm\varepsilon$, and per-layer feedback matrices $B_\ell$ that stand in for $W_\ell^\top$ are learned once per update from random probes sent through the same $W_\ell$.

**Measuring neuron slopes.** To carry the error through a nonlinear layer, the backward pass needs the derivative of $\phi_\ell$ at the current input $z_\ell$. Rather than deriving this from a mathematical model of the neuron, we measure it on the network itself using central finite differences, a classical primitive whose error is $O(\varepsilon^2)$ and which requires only forward evaluations (Moré & Wild, 2012). For each layer, we choose a perturbation magnitude $\varepsilon > 0$ and evaluate $\phi_\ell$ at $z_\ell \pm \varepsilon\mathbf{1}$, where $\mathbf{1}$ is the all-ones vector. The measured slopes are

$$s_\ell = \frac{\phi_\ell(z_\ell + \varepsilon\mathbf{1}) - \phi_\ell(z_\ell - \varepsilon\mathbf{1})}{2\varepsilon}. \tag{2}$$

Because $\phi_\ell$ acts elementwise, these two additional evaluations measure the slopes of all neurons in the layer at once, independent of layer width.

**Learning feedback matrices.** To carry the error through a linear layer, BP multiplies the incoming error by $W_\ell^\top$. Copying $W_\ell$ into the backward pathway is not available on physical substrates or in biology (the *weight-transport* problem). The idea of instead learning a separate matrix $B_\ell$ that plays the role of $W_\ell^\top$ from locally available signals goes back to Kolen & Pollack (1994). A more recent instantiation is the weight mirror of Akrout et al. (2019), which drives the forward-path neurons of a chosen layer to fire noisily in a dedicated "mirror mode" and learns $B$ from the resulting correlations across $W$, alternating with normal forward-path operation. We take a different route: we inject independent random probes directly at the *input* of the linear map $W_\ell$ and read them out at its output, decoupling feedback learning from the forward-path dynamics of the surrounding network. This

lets a single normalized-correlation update recover $W_\ell^\top$ exactly in expectation, run alongside the forward pass without switching operating modes, and reuse without change on any layer that admits forward probing, from Transformer weight matrices to spiking synapses (Section 2.3). Concretely, for $m$ independent Gaussian probes $\xi_{\ell,r} \sim \mathcal{N}(0, I)$, we measure the layer's forward response $q_{\ell,r} = W_\ell \xi_{\ell,r}$ and update each feedback entry as

$$(B_\ell)_{ji} \leftarrow (1-\rho)(B_\ell)_{ji} + \rho \, \frac{\sum_{r=1}^{m} (\xi_{\ell,r})_j (q_{\ell,r})_i}{\sum_{r=1}^{m} (\xi_{\ell,r})_j^2}, \tag{3}$$

where $\rho \in (0, 1)$ is the feedback learning rate. The per-coordinate normalization by $\sum_r (\xi_{\ell,r})_j^2$ makes the update's target $W_\ell^\top$ itself, independent of probe amplitude, which is what later allows Eq. (6) to reuse the same rule with spike-difference probes whose amplitudes are not controlled. Gaussian probes are the simplest choice that satisfies the zero-mean, coordinate-independent condition Eq. (3) requires; any distribution meeting that condition would work, as the spiking case demonstrates. Because $W_\ell$ changes with every optimizer step, $B_\ell$ has to track it: we refresh it after every weight update, and run 20 refreshes before training so the first few steps do not train under a random feedback pathway.

**Backward pass and weight update.** Starting from the loss derivative $v_L = \nabla_{a_L} \mathcal{L}$, we propagate error signals backward through the layers, computing

$$g_\ell = s_\ell \odot v_\ell, \qquad v_{\ell-1} = B_\ell g_\ell \quad (\ell > 1). \tag{4}$$

Here $v_\ell$ is the sensitivity of the loss to the layer's output $a_\ell$, and $g_\ell$ the sensitivity to its input $z_\ell$; the measured slopes carry the error from $v_\ell$ to $g_\ell$ through the nonlinear layer, and the learned feedback matrix carries it from $g_\ell$ to $v_{\ell-1}$ through the linear layer. In BP these are exactly $\partial \mathcal{L}/\partial a_\ell$ and $\partial \mathcal{L}/\partial z_\ell$; PerProp produces their measurement-and-feedback approximations. With learning rate $\eta > 0$, we then update each weight matrix from its local presynaptic activity and postsynaptic error,

$$W_\ell \leftarrow W_\ell - \eta \, g_\ell a_{\ell-1}^\top. \tag{5}$$

The weight-gradient estimate $g_\ell a_{\ell-1}^\top$ can also be supplied to another optimizer.

### 2.3 Perturbation Propagation for spiking neural networks

Spiking neural networks (SNNs) use neurons that integrate incoming currents over time and transmit spikes to connected neurons when their membrane potential exceeds a firing threshold. Their event-driven communication and native compatibility with analog neuromorphic hardware make them a leading substrate for biologically plausible, low-power computation. Because spike generation is a threshold step whose derivative is zero almost everywhere and undefined at the threshold, exact gradients carry no learning signal, and standard training relies on hand-designed surrogate derivatives (Neftci et al., 2019). PerProp extends to this setting with essentially no change to its design: replacing the local slope and probe responses of Section 2.2 with their spike-response counterparts preserves the normalized-correlation update while operating directly on spiking signals, without requiring a surrogate derivative for each neuron model. For a synaptic map $W$, paired neuron rollouts with positive and negative current perturbations produce spike differences $\Delta s_r = (s_r^+ - s_r^-)/(2\varepsilon)$ and forward current responses $\Delta I_r = W \Delta s_r$. Substituting these signals for the Gaussian probes and their responses in equation (3) gives

$$B_{ji} \leftarrow (1-\rho) B_{ji} + \rho \, \frac{\sum_r (\Delta s_r)_j (\Delta I_r)_i}{\sum_r (\Delta s_r)_j^2}, \tag{6}$$

where $r$ indexes examples and time steps; feedback entries associated with negligible spike-difference power remain unchanged. Local neuron-state sensitivities are measured through coordinate perturbations. Appendix C.3 details the measurement schedule and establishes the conditional unbiasedness of the feedback target. This same locality is what later allows PerProp to run end-to-end on the BrainScaleS-2 neuromorphic chip (Section 3.4).

## 3 Experiments and Results

Our experiments address three questions. How does PerProp compare with BP and other alternatives in scalability, predictive performance, and training time (§3.1–3.2)? Can it train non-differentiable networks, including spiking neural networks, with a common learning rule (§3.3)? Can it learn directly on neuromorphic hardware, and what energy savings do we project for an analog crossbar network implementation (§3.4)? BP provides a reference for performance and scalability. Hybrid

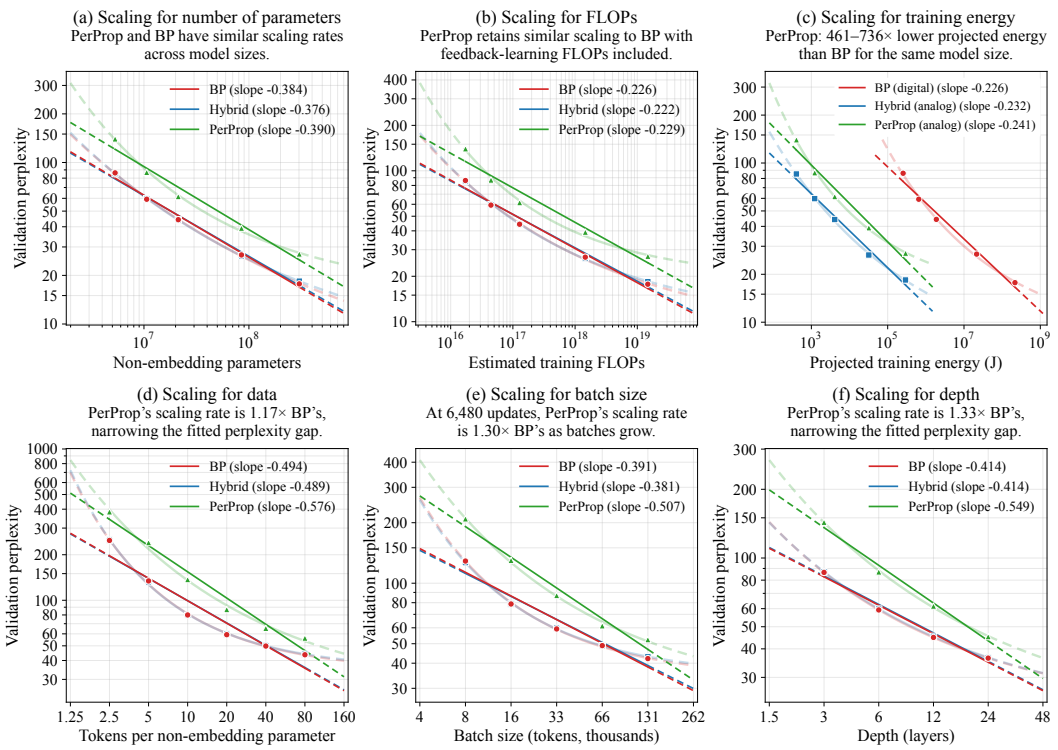


Figure 2: Scaling on FineWeb-Edu. Each point uses one seed; perplexity is computed from the mean held-out loss over the final 10% of updates (at least 500). Panel (e) fixes 6,480 updates, and panel (b) includes feedback-learning FLOPs. Faint curves show nonlinear loss fits; dark lines show log–log fits. Dashed segments indicate extrapolation. Panel (c) uses the energy projection from Section 3.4.

serves as an ablation of feedback learning by replacing PerProp's learned feedback matrices with the transposed forward weights while retaining its measured neuron slopes.

### 3.1 Learning scalability

In this section, we examine how Perturbation Propagation scales relative to BP and Hybrid when pretraining Transformer models on FineWeb-Edu. Preserving the performance gains from larger models and greater training budgets is essential for extending a backpropagation alternative to large-scale pretraining. To assess scalability comprehensively, we evaluate 17 configurations spanning training tokens, parameter count, batch size, depth, estimated training FLOPs, and projected training energy.

PerProp scales faster than backpropagation with increasing training data, batch size, and depth, narrowing the validation-perplexity gap across these sweeps (Figure 2(d–f)). The data sweep fixes architecture and batch size; the batch-size sweep fixes the number of optimizer updates, so larger batches also process more tokens.

For depth and model-size scaling, we allocate 20 training tokens per non-embedding parameter, following Chinchilla's proportional scaling of model size and training data (Hoffmann et al., 2022). Across models up to 0.4 billion parameters, PerProp achieves fitted scaling rates comparable to BP and Hybrid, and this trend persists against estimated training FLOPs including feedback-learning costs (Figure 2(a,b)).

### 3.2 Comparison with other backpropagation alternatives

In this section, we compare PerProp with other backpropagation alternatives to evaluate the trade-off between predictive performance and training time.

We train 24-layer GPT (23.2M parameters) and Mamba-2 (14.6M parameters) models on WikiText-103 (Merity et al., 2016) (Table 1; settings in Appendix E.2). For out-of-domain evaluation, we use the same checkpoints without fine-tuning on all 21,990 validation stories from TinyStories, a dataset of short synthetic stories (Eldan & Li, 2023). We retain the WikiText tokenizer and use 128-token windows with a stride of 64. PerProp outperforms the feedback-alignment, DFA, and weight- or

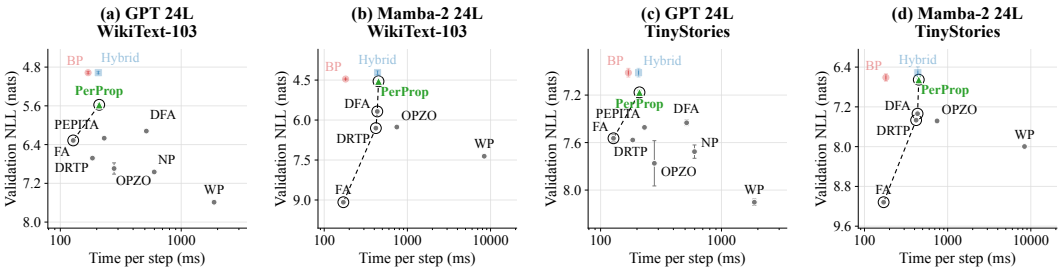


Figure 3: Pareto frontiers of performance and training time per step on an RTX A6000. BP and Hybrid are faded references excluded from the frontier because they use transposed forward weights.

| | WikiText-103: in-domain NLL | | TinyStories: out-of-domain NLL | |
|---|---|---|---|---|
| Learning rule | GPT 24L | Mamba-2 24L | GPT 24L | Mamba-2 24L |
| BP | 4.915 ± 0.022 | 4.461 ± 0.017 | 7.011 ± 0.036 | 6.609 ± 0.075 |
| Hybrid | 4.915 ± 0.022 | 4.222 ± 0.070 | 7.011 ± 0.036 | 6.512 ± 0.117 |
| **PerProp** | **5.579 ± 0.023** | **4.546 ± 0.019** | **7.177 ± 0.019** | **6.653 ± 0.059** |
| Feedback alignment | 6.314 ± 0.043 | 9.084 ± 0.057 | 7.564 ± 0.029 | 9.118 ± 0.032 |
| Direct feedback alignment | 6.121 ± 0.009 | 5.677 ± 0.010 | 7.433 ± 0.023 | 7.337 ± 0.006 |
| DRTP | 6.679 ± 0.009 | 6.303 ± 0.011 | 7.579 ± 0.006 | 7.467 ± 0.012 |
| PEPITA | 6.268 ± 0.008 | div. | 7.472 ± 0.012 | div. |
| Weight perturbation / MeZO | 7.590 ± 0.011 | 7.357 ± 0.009 | 8.102 ± 0.027 | 7.997 ± 0.025 |
| Node perturbation | 6.965 ± 0.013 | div. | 7.676 ± 0.056 | div. |
| OPZO | 6.894 ± 0.116† | 6.262 ± 0.010 | 7.775 ± 0.191† | 7.483 ± 0.046 |

Table 1: Validation NLL (nats, ↓) on WikiText-103 and TinyStories for models trained on WikiText-103. Entries report mean ± sample standard deviation over three seeds. †: two completed seeds; the third diverged. "div.": training divergence.

node-perturbation entries in Table 1. CIFAR-10 results for ResMLP-16 and ConvNeXt V2 are in Appendix G.3.

We find that weight and node perturbation approaches achieve poorer performance in these experiments compared to PerProp, consistent with the dimension-dependent variance of random-direction gradient estimates (Oripov et al., 2025). PerProp measures elementwise neuron slopes in parallel with two forward evaluations per nonlinear layer, avoiding both nonlinear coordinate sweeps and random-direction estimation of the full update. Separate forward probes learn the linear feedback operators.

Overall, PerProp is the strongest backpropagation alternative in this comparison: it attains the lowest in-domain and out-of-distribution validation loss on both GPT and Mamba-2, and remains on the accuracy-time Pareto frontier against every alternative that avoids weight transport.

### 3.3 Unified training of non-differentiable units

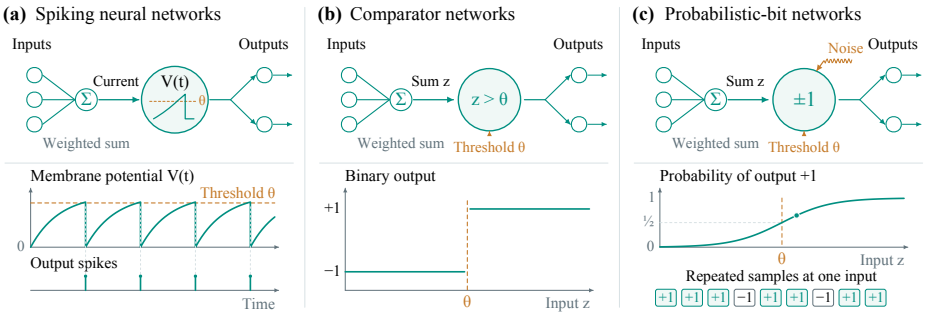


Figure 4: Representative neurons with incoming and outgoing connections (top) and illustrative responses (bottom) for (a) spiking, (b) comparator, and (c) probabilistic-bit networks.

In this section, we evaluate PerProp as a unified approach to training comparator, probabilistic-bit, and spiking networks from measured neuron responses. Gradient-based training of non-differentiable networks commonly relies on prescribed surrogate derivatives (Bengio et al., 2013; Neftci et al., 2019). On neuromorphic hardware, implementations can require an external processor to compute the backward pass (Cramer et al., 2022a). Direct on-chip training could reduce data transfers to external processors and avoid ANN-to-SNN conversion, while allowing networks to adapt to hardware imperfections (Ostrau et al., 2020). Figure 4 illustrates their network structures and neuron responses.

**Spiking neural networks** We evaluate the spiking formulation of PerProp (Section 2.3) in simulation on rate-coded MNIST using leaky, quadratic, and exponential integrate-and-fire neurons, as well as Izhikevich neurons. Each network has two hidden layers of 256 neurons and is unrolled for 25 time steps. For temporal classification, we use Spiking Heidelberg Digits (SHD) (Cramer et al., 2022b) with one recurrent layer of 256 adaptive leaky integrate-and-fire neurons and 100 time bins. We compare PerProp with surrogate backpropagation through time (BPTT), reporting test accuracy across three seeds. Table 2 reports the PerProp and surrogate-BPTT results. PerProp matches surrogate BPTT across four neuron models on MNIST and remains within noise on SHD, learning from measured neuron responses rather than hand-designed surrogates.

Table 2: SNN test accuracy (%), mean ± sample standard deviation over three seeds. BPTT uses surrogate gradients. EIF uses validation-selected settings for both methods.

| Setting | PerProp | BPTT |
|---|---|---|
| *MNIST* | | |
| LIF | $97.04 \pm 0.33$ | $97.25 \pm 0.17$ |
| QIF | $97.09 \pm 0.11$ | $97.23 \pm 0.16$ |
| EIF | $97.04 \pm 0.09$ | $96.54 \pm 0.32$ |
| Izhikevich | $96.89 \pm 0.15$ | $97.22 \pm 0.05$ |
| SHD (ALIF) | $86.44 \pm 0.50$ | $86.90 \pm 0.93$ |

**Comparator networks and probabilistic-bit networks** Here we show the trainability of other non-differentiable networks. In comparator networks, each neuron combines incoming signals into a weighted sum and compares this scalar with a reference threshold, producing a binary activation of $+1$ or $-1$. Probabilistic-bit networks use the same two possible outputs, with each neuron choosing randomly between them according to a probability determined by its input. Both use the PerProp training rule from Section 2.2, with measured nonlinear responses and learned feedback.

We construct both networks by replacing the GELU activations in ResMLP-16 with the corresponding binary activations. We train on CIFAR-10 with fixed per-neuron thresholds $\theta_j \sim \mathcal{N}(0, 0.3^2)$. The BP baselines use a straight-through estimator for comparators and an expectation-based surrogate for probabilistic bits, both assuming a nominal threshold of zero. The paired probabilistic-bit measurements share the same noise realization. Table 3 reports the results over three seeds. PerProp outperforms surrogate BP on both comparator ($+1.19$ pp) and probabilistic-bit ($+1.18$ pp) networks, using measured neuron responses in place of hand-designed surrogates.

Overall, a single PerProp rule trains spiking, comparator, and probabilistic-bit networks from measured neuron responses alone, matching or exceeding surrogate-gradient training on every architecture we evaluate without requiring a hand-designed surrogate for each neuron type.

Table 3: CIFAR-10 test accuracy (%) of comparator and probabilistic-bit networks, reported as mean ± sample standard deviation over three seeds.

| Network | PerProp | Hybrid | Surrogate BP |
|---|---|---|---|
| Comparator network | $48.59 \pm 0.57$ | $48.58 \pm 0.28$ | $47.40 \pm 0.32$ |
| Probabilistic-bit network | $48.35 \pm 0.16$ | $48.49 \pm 0.60$ | $47.17 \pm 0.31$ |

### 3.4 Hardware in the loop

In this section, we evaluate whether PerProp can train neuromorphic chips and analog neural networks under hardware imperfections and estimate its training and inference energy on the target analog hardware

**BrainScaleS-2** We train a spiking network directly on BrainScaleS-2, a neuromorphic chip whose analog circuits implement neurons and synapses (Pehle et al., 2022). Its on-chip digital plasticity processing unit (PPU) executes a hardware-specific implementation of PerProp, using measured perturbation responses to learn feedback connections, propagate error signals, and update synaptic weights. A host computer supplies training examples and labels and coordinates execution. Hardware architecture and implementation details are provided in Appendix I.

We evaluate on Yin–Yang, a three-class classification task based on regions of a two-dimensional pattern, developed for studying learning in spiking networks (Kriener et al., 2021). Our network uses five encoded input channels, 32 hidden spiking neurons, and three output neurons, with both synaptic layers trained on-chip.

On BrainScaleS-2 we benchmark against R-STDP, the only established fully on-chip learning rule with a published task-training implementation on this platform (Wunderlich et al., 2019), and improve Yin–Yang accuracy by **25** percentage points. Both methods execute their weight updates on the PPU and train the same network for five epochs. PerProp and R-STDP use the same data splits and paired initial weights. We select checkpoints by validation accuracy and report performance on 1,000 held-out test examples. PerProp achieves $62.1 \pm 1.8\%$ test accuracy versus $36.9 \pm 1.8\%$ for R-STDP across three seeds, against a uniform-guessing baseline of $33.3\%$ on this three-class task.

These results demonstrate that PerProp can learn both hidden and output connections from physical neuron responses, with feedback learning, error propagation, and weight updates executed on-chip.

**Crossbar networks** We simulate the crossbar architecture illustrated in Figure 5, where weights are represented by programmable electrical conductances: input voltages produce output currents that represent weighted sums. Neuron circuits apply nonlinear responses between successive arrays.

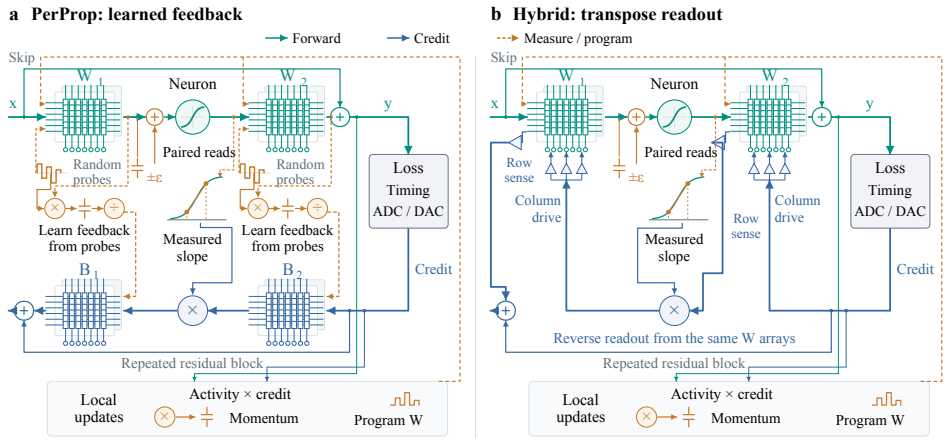


Figure 5: Proposed crossbar architectures. (a) PerProp learns separate feedback arrays $B$ from forward probes. (b) Hybrid drives columns and senses rows of the same $W$ arrays to apply $W^\top$. Both measure neuron slopes with paired forward evaluations.

In the proposed PerProp architecture (Figure 5(a)), separate crossbar arrays store the feedback matrices $B$ (blue). Random probe voltages are applied to each forward array $W$, and the resulting output currents are measured. Multipliers and averaging circuits accumulate the probe–response products, normalize them by the accumulated squared probe amplitudes, and update the feedback weights according to Eq. (3). For neuron-slope measurement, a sample-and-hold circuit retains each neuron's input while small positive and negative offsets are applied. The difference between the two responses, divided by $2\varepsilon$, gives the measured slope. This slope multiplies the incoming error signal, which then passes through the corresponding $B$ array to the preceding layer. Each forward weight receives a local update formed from its input activation and output error signal. Many crossbar architectures also support direct transpose multiplication by driving their columns and sensing currents along their rows. We therefore include Hybrid (Figure 5(b)), which uses these additional drive and sensing circuits to apply $W^\top g$ through the original weight arrays. It retains the same neuron measurements and local updates while eliminating the separate feedback arrays and their learning circuitry.

We compare PerProp and Hybrid with an oracle BP baseline that has access to the exact neuron derivatives and weight transposes in the simulator. All three methods train an eight-block residual MLP of width 128 on CIFAR-10 for 12,000 updates, using three paired seeds. The simulation includes 7-bit programming of the forward and feedback weights, 8-bit input conversion, programming and readout noise, and variation in individual neuron responses.

PerProp comes within 1.78 pp of the oracle BP baseline ($42.34 \pm 0.77\%$ vs $44.12 \pm 0.07\%$), despite operating under 7-bit weight programming, 8-bit input quantization, and per-neuron gain/offset variation that the oracle sees through perfectly.

We estimate energy by combining operation counts with component-level MAC energies for the proposed architectures, comparing training against a historical V100 reference (Patterson et al., 2021) and inference against the H100 SXM's dense BF16 peak throughput (NVIDIA, n.d.). Appendix K details the accounting protocol (Wang & Achour, 2026), per-component budgets, and sensitivity ranges.

Table 4 extrapolates energy across four network sizes and a range of published and target MAC costs. PerProp projects orders-of-magnitude energy gains over digital backpropagation for both training and inference, and these gains widen with network size: at the 2.17-billion-weight scale, the measured 17.1 fJ/MAC reference already yields three-order-of-magnitude training-energy savings, with further gains available as analog MAC costs approach their design and ideal targets.

Table 4: Projected energy across analog network sizes at batch 2,048. Each cell gives energy per example and the corresponding digital-to-analog energy ratio.

| MAC cost (fJ) | 0.66M weights | 9.97M weights | 140.5M weights | 2.17B weights |
|---|---|---|---|---|
| *Training: μJ/example (gain over historical GPU training)* | | | | |
| 204.9 | 0.300 (155×) | 4.908 (170×) | 66.960 (182×) | 1025.248 (186×) |
| 167.9 | 0.261 (178×) | 4.142 (201×) | 55.511 (220×) | 845.299 (226×) |
| 48.9 | 0.137 (340×) | 1.677 (497×) | 18.668 (654×) | 266.232 (717×) |
| 17.1 | 0.103 (449×) | 1.018 (818×) | 8.828 (1,383×) | 111.570 (1,711×) |
| 9.5 (design) | 0.095 (486×) | 0.862 (967×) | 6.485 (1,883×) | 74.744 (2,554×) |
| 4.0 (target) | 0.090 (518×) | 0.747 (1,115×) | 4.775 (2,557×) | 47.873 (3,988×) |
| 1.0 (target) | 0.086 (536×) | 0.685 (1,216×) | 3.847 (3,174×) | 33.280 (5,736×) |
| *Inference: μJ/example (gain over H100 BF16 peak reference)* | | | | |
| 204.9 | 0.170 (5×) | 2.414 (6×) | 32.345 (6×) | 492.427 (6×) |
| 167.9 | 0.143 (6×) | 2.008 (7×) | 26.627 (7×) | 404.016 (8×) |
| 48.9 | 0.057 (16×) | 0.703 (20×) | 8.225 (24×) | 119.514 (25×) |
| 17.1 | 0.035 (27×) | 0.355 (39×) | 3.311 (59×) | 43.527 (70×) |
| 9.5 (design) | 0.029 (32×) | 0.272 (51×) | 2.140 (92×) | 25.434 (120×) |
| 4.0 (target) | 0.025 (37×) | 0.211 (66×) | 1.286 (153×) | 12.232 (249×) |
| 1.0 (target) | 0.023 (40×) | 0.178 (78×) | 0.823 (239×) | 5.062 (601×) |

Training ratios use 29.4 pJ/MAC from historical V100 training; inference ratios use the 1.4 pJ/MAC H100 reference. Correlation accumulation is 0.1 pJ/product. The 4 and 1 fJ rows are ideal targets; 9.5 fJ is a published design projection. Component budgets and the 10% routing allowance follow Appendix K.

Overall, PerProp learns on analog hardware end-to-end: it trains on the BrainScaleS-2 neuromorphic chip with all learning steps executed on-chip, tolerates the quantization, noise, and device variation of a simulated crossbar while staying close to an oracle BP baseline, and projects orders-of-magnitude energy gains over GPU backpropagation that widen with network size.

## 4 Conclusion

We introduced PerProp, a training algorithm that keeps the forward-backward structure of backpropagation but replaces its two biologically implausible operations with local measurements: neuron derivatives with slopes measured by small forward perturbations, and weight transposes with feedback matrices learned from random probes. Across Transformer pretraining on FineWeb-Edu, language modeling on WikiText-103 and TinyStories, non-differentiable comparator, probabilistic-bit, and spiking networks, and analog neuromorphic hardware, a single PerProp rule scales at rates comparable to or steeper than backpropagation, outperforms the backpropagation alternatives we evaluate, matches or exceeds hand-designed surrogates on non-differentiable units, and trains end-to-end on the BrainScaleS-2 chip.

Because every step of PerProp runs on local measurements of the network itself, the algorithm is directly compatible with physical neural networks whose analog dynamics implement the forward pass. Our BrainScaleS-2 experiments show that this compatibility holds in practice on today's neuromorphic hardware; our crossbar simulations project that, as analog MAC costs approach their design and ideal targets, the same rule could deliver orders-of-magnitude energy gains over digital backpropagation, widening with network size. Future work includes extending the scaling study beyond 0.4 billion parameters, validating the projected energy savings on integrated analog hardware, applying PerProp to larger networks and more complex tasks on BrainScaleS-2, and developing a quantitative theory of its scaling behavior. Taken together, these results position PerProp as a concrete step toward the much more energy-efficient physical neural networks that a scalable, biologically plausible training algorithm makes possible.

### AI USE STATEMENT

In this work, we used generative AI tools (Claude Code and Codex, as coding agents) for writing and debugging experiment code, orchestrating and monitoring training runs. All experimental designs, claims and conclusions are the authors' own; every number reported was produced by the released code and checked by the authors. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work studies training rules for physical and simulated learning systems; it uses public datasets (CIFAR-10, WikiText-103, TinyStories, FineWeb-Edu, MNIST, the Spiking Heidelberg Digits, Yin–Yang) and involves no human subjects or personal data. The hardware evaluation combines crossbar simulations anchored to published silicon with experiments on a physical BrainScaleS-2 chip.

### REPRODUCIBILITY STATEMENT

Every experiment is run from the released code with the exact commands recorded in the results files; the appendix states the settings, seeds and access-cost accounting for each table, the pre-registered predictions written before the runs.

### ACKNOWLEDGMENTS

## REFERENCES

Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Syme, Dmitry Wood, and Philip Torr. Gradients without backpropagation, 2022.

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*, 11:3625, 2020.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

Sansheng Cao, Zhengyu Ma, and Yonghong Tian. Hierarchical zero-order optimization for deep neural networks, 2026. URL https://arxiv.org/abs/2602.10607.

Yiming Chen, Yuan Zhang, Liyuan Cao, Kun Yuan, and Zaiwen Wen. Enhancing zeroth-order fine-tuning for language models with low-rank structures. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/9ccc9d814d3dee4750debaf23061e733-Paper-Conference.pdf.

Andrew Clark, Jack Moursounidis, Osmaan Rasouli, William Gan, Cooper Doyle, and Anna Leontjeva. BOND: License to train with black-box functions, 2026. URL https://arxiv.org/abs/2510.25074.

Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Johannes Schemmel, and Friedemann Zenke. Surrogate gradients for analog neuromorphic computing. *Proceedings of the National Academy of Sciences*, 119(4):e2109194119, 2022a. doi: 10.1073/pnas.2109194119. URL https://doi.org/10.1073/pnas.2109194119.

Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2744–2757, 2022b.

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, et al. Advancing mathematics by guiding human intuition with ai. *Nature*, 600:70–74, 2021. doi: 10.1038/s41586-021-04086-x.

Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 2022. URL https://proceedings.mlr.press/v162/dellaferrera22a.html.

Sam Dillavou, Marcelo Guzman, Andrea J. Liu, and Douglas J. Durian. Understanding and embracing imperfection in physical learning networks, 2025.

Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent english?, 2023. URL https://arxiv.org/abs/2305.07759.

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*, 15:629892, 2021. doi: 10.3389/fnins.2021.629892.

Tayfun Gokmen and Yurii Vlasov. Acceleration of deep neural network training with resistive cross-point devices: Design considerations. *Frontiers in Neuroscience*, 10:333, 2016. doi: 10.3389/fnins.2016.00333.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030. Curran Associates, Inc., 2022. doi: 10.52202/068431-2176. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.

Wei-Cheng Huang, Hsin-Ta Chien, Soon-Jyh Chang, Jia-Hao Hu, and Chih-Hung Kuo. A fully analog computing-in-memory macro with INT8-MAC operations for edge-AI device. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 16(2):239–251, 2026. doi: 10.1109/JETCAS.2026.3657011.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589, 2021. doi: 10.1038/s41586-021-03819-2.

Win-San Khwa, Tai-Hao Wen, Hung-Hsi Hsu, Wei-Hsing Huang, Yu-Chen Chang, Ting-Chien Chiu, Zhao-En Ke, Yu-Hsiang Chin, Hua-Jin Wen, Wei-Ting Hsu, et al. A mixed-precision memristor and sram compute-in-memory ai processor. *Nature*, 639:617–623, 2025. doi: 10.1038/s41586-025-08639-2.

John F Kolen and Jordan B Pollack. Backpropagation without weight transport. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pp. 1375–1380, 1994.

Laura Kriener, Julian Göltz, and Mihai A. Petrovici. The Yin-Yang dataset, 2021. URL https://arxiv.org/abs/2102.08211.

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

Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *Frontiers in Neuroscience*, 15:633674, 2021. doi: 10.3389/fnins.2021.633674.

Manuel Le Gallo, Riduan Khaddam-Aljameh, Milos Stanisavljevic, Athanasios Vasilopoulos, Benedikt Kersting, Martino Dazzi, Geethan Karunaratne, Matthias Brändli, Abhairaj Singh, Silvia M. Mueller, et al. A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. *Nature Electronics*, 6:680–693, 2023. doi: 10.1038/s41928-023-01010-1.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, Antoine Biard, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pp. 498–515, 2015. doi: 10.1007/978-3-319-23528-8_31.

William B. Levy and Victoria G. Calvert. Communication consumes 35 times more energy than computation in the human cortex, but both costs are needed to predict synapse number. *Proceedings of the National Academy of Sciences*, 118(18):e2008173118, 2021. doi: 10.1073/pnas.2008173118. URL https://doi.org/10.1073/pnas.2008173118.

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: 13276, 2016.

Yong Liu, Zirui Zhu, Chaoyu Gong, Minhao Cheng, Cho-Jui Hsieh, and Yang You. Sparse MeZO: Less parameters for better performance in zeroth-order LLM fine-tuning. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1e5c2efbddc02c1d971e2f19ccdb07d0-Paper-Conference.pdf.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason Dovonon, Xingyu Lyu, et al. Fine-tuning language models with just forward passes. In *NeurIPS*, 2023.

Amil Merchant, Simon Batzner, Samuel S. Schoenholz, Muratahan Aykol, Gowoon Cheon, Ekin Dogus Cubuk, et al. Scaling deep learning for materials discovery. *Nature*, 624:80–85, 2023. doi: 10.1038/s41586-023-06735-9.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Ali Momeni, Babak Rahmani, Benjamin Scellier, Logan G. Wright, Peter L. McMahon, Clara C. Wanjura, Yuhang Li, Anas Skalli, Natalia G. Berloff, Tatsuhiro Onodera, et al. Training of physical neural networks. *Nature*, 2025. doi: 10.1038/s41586-025-09384-2.

Jorge J. Moré and Stefan M. Wild. Estimating derivatives of noisy simulations. *ACM Transactions on Mathematical Software*, 38(3):19:1–19:21, 2012. doi: 10.1145/2168773.2168777.

E. O. Neftci, H. Mostafa, and F. Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. doi: 10.1109/MSP.2019.2931595.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

NVIDIA. NVIDIA H100 GPU: Product specifications. https://www.nvidia.com/en-us/data-center/h100/, n.d. Accessed September 22, 2026.

Bakhrom G. Oripov, Andrew Dienstfrey, Adam N. McCaughan, and Sonia M. Buckley. Scaling of hardware-compatible perturbative training algorithms, 2025. URL https://arxiv.org/abs/2501.15403.

Christoph Ostrau, Jonas Homburg, Christian Klarhorst, Michael Thies, and Ulrich Rückert. Benchmarking deep spiking neural networks on neuromorphic hardware. In *Artificial Neural Networks and Machine Learning – ICANN 2020*, pp. 610–621. Springer International Publishing, 2020. ISBN 9783030616168. doi: 10.1007/978-3-030-61616-8_49. URL http://dx.doi.org/10.1007/978-3-030-61616-8_49.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021.

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. doi: 10.3389/fnins.2022.795876. URL https://doi.org/10.3389/fnins.2022.795876.

Fernando M. Quintana, Fernando Perez-Peña, Pedro L. Galindo, Emre O. Neftci, Elisabetta Chicca, and Lyes Khacef. ETLP: Event-based three-factor local plasticity for online learning with neuromorphic hardware. *Neuromorphic Computing and Engineering*, 4(3):034006, 2024. doi: 10.1088/2634-4386/ad6733.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017. doi: 10.3389/fncom.2017.00024.

Matan Stern, Daniel Hexner, Jason W. Rocks, and Alice J. Liu. Supervised learning in physical networks: From machine learning to learning machines. *Physical Review X*, 11(2):021045, 2021. doi: 10.1103/PhysRevX.11.021045.

Yu-Neng Wang and Sara Achour. Generative models on analog hardware with dynamics, 2026.

Logan G. Wright, Tatsuhiro Onodera, Martin M. Stein, Tianyu Wang, Darren T. Schachter, Zoey Hu, and Peter L. McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, January 2022. doi: 10.1038/s41586-021-04223-6. URL https://www.nature.com/articles/s41586-021-04223-6.

Timo Wunderlich, Akos F. Kungl, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht, Korbinian Schreiber, David Stöckel, Christian Pehle, Sebastian Billaudelle, Gerd Kiene, Christian Mauch, Johannes Schemmel, Karlheinz Meier, and Mihai A. Petrovici. Demonstrating advantages of neuromorphic computation: A pilot study. *Frontiers in Neuroscience*, 13:260, 2019. doi: 10.3389/fnins.2019.00260. URL https://doi.org/10.3389/fnins.2019.00260.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online pseudo-zeroth-order training of neuromorphic spiking neural networks, 2024.

Cheng-Xin Xue, Je-Min Hung, Hui-Yao Kao, Yen-Hsiang Huang, Sheng-Po Huang, Fu-Chun Chang, Peng Chen, Ta-Wei Liu, Chuan-Jia Jhang, Chin-I Su, Win-San Khwa, Chung-Chuan Lo, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, Yu-Der Chih, Tsung-Yung Jonathan Chang, and Meng-Fan Chang. A 22nm 4mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7 TOPS/W for tiny AI edge devices. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 245–247, 2021. doi: 10.1109/ISSCC42613.2021.9365769.

Pengyun Yue, Xuanlin Yang, Mingqing Xiao, and Zhouchen Lin. PseuZO: Pseudo-zeroth-order algorithm for training deep neural networks. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=tM4cHBD7kD.

# A Background and related work

This appendix introduces the physical and biological learning settings used in the paper and relates the design to earlier learning rules. The distinctions concern what signals a method needs, how it assigns credit, and where its computations execute.

## A.1 Physical and neuromorphic neural networks

A physical neural network uses the response of a physical system to perform part of a neural computation. For example, electrical conductances can multiply input voltages and sum the resulting currents, while optical propagation or electronic neuron circuits provide other transformations. Training changes controllable parameters so that the measured system produces the desired outputs. A digital simulation of these responses is a model of a PNN; the physical system itself performs the computation only when inputs are applied and outputs measured. Physics-aware training combines physical forward responses with a differentiable digital model for the backward calculation (Wright et al., 2022).

Neuromorphic hardware implements neural and synaptic operations in dedicated circuits. Its neurons may be digital or analog. BrainScaleS-2 uses analog membrane and synaptic dynamics, together with digital spike routing and an on-chip processor for plasticity (Pehle et al., 2022). It is therefore both a neuromorphic platform and a physical implementation of spiking-neuron dynamics. In this paper, the GPU SNN experiments establish trainability for several neuron responses; the BrainScaleS-2 experiment establishes execution of learning on the physical chip; and the crossbar simulation studies a separate proposed analog architecture.

The opportunity for energy savings comes from performing useful operations directly in the substrate and reducing data movement. The total also includes sensing, conversion, communication, and weight updates. We therefore evaluate the complete accounted learning schedule when projecting crossbar energy, rather than multiplying inference efficiency by an assumed training speedup.

## A.2 Biological plausibility and credit assignment

In standard backpropagation, the error at an earlier layer depends on the transpose of a later layer's weights and on local derivatives. A separate biological feedback pathway does not automatically have matching synaptic strengths, motivating the weight-transport problem. Feedback alignment replaces exact transposed feedback with fixed random connections (Lillicrap et al., 2016); direct feedback alignment projects the output error directly to hidden layers (Nøkland, 2016). Direct random target projection uses label projections as teaching signals (Frenkel et al., 2021).

There exists a few explorations on learning feedback. Kolen–Pollack updates and weight mirroring align feedback with forward connections through learning (Kolen & Pollack, 1994; Akrout et al., 2019). Weight mirroring already demonstrated deep-network learning on ImageNet. Our method builds on the feasibility of learned feedback while combining normalized input–response correlations with measured neuron sensitivities. It retains a backward path from the global loss and supplies local weight increments along that path. The reported slope measurements and feedback measurements use separate forward evaluations.

The three-factor form separates presynaptic activity, postsynaptic sensitivity or eligibility, and a teaching signal. It appears in several biologically motivated approaches. E-prop combines local eligibility traces with learning signals to approximate temporal credit assignment (Bellec et al., 2020). Reward-modulated STDP instead modulates spike-timing eligibility with reward and has an on-chip BrainScaleS-2 task demonstration (Wunderlich et al., 2019). PerProp obtains its sensitivity factor from paired responses and its hidden teaching signals from learned feedback. On-chip implementation establishes that these signals can be formed with the selected hardware primitives; it is not evidence that biological brains implement this particular rule.

## A.3 Forward differentiation, perturbations, and local objectives

Forward-mode automatic differentiation propagates a directional derivative along a forward computation. The forward-gradient method uses such a directional derivative to form an unbiased gradient estimate (Baydin et al., 2022). Finite-difference methods instead observe changes in function values, making them applicable when analytical derivatives are unavailable. Weight perturbation probes parameter directions; node perturbation probes internal activity. Their cost depends on the number and structure of probes and on the information exposed by each measurement.

PerProp exploits elementwise neuron responses to measure all local scalar slopes with two vector evaluations. It then reuses learned local feedback operators across the examples in a training batch.

This differs from estimating the complete parameter gradient from a scalar loss difference. It also differs from identifying the entire linear map by a coordinate sweep: the experiments use a finite stochastic probe batch and retain a noisy feedback estimate. Appendix D gives the corresponding fixed-map moments and convergence conditions.

Other approaches change the learning signal or objective. Difference target propagation learns approximate inverses and propagates targets (Lee et al., 2015). Forward–forward trains local goodness objectives on positive and negative examples (Hinton, 2022); PEPITA uses differences between nominal and error-modulated forward activity (Dellaferrera & Kreiman, 2022). Equilibrium propagation obtains updates from the difference between free and weakly nudged equilibria of an energy-based network (Scellier & Bengio, 2017). These methods make different requirements on network dynamics, state storage, and available signals. Appendix G specifies the implementations used in our comparisons, including their architecture-specific adaptations.

### A.4 Non-differentiable neuron responses

A spiking neuron integrates input over time and emits a discrete event when its membrane crosses a threshold. A comparator neuron outputs one of two values according to whether a weighted sum exceeds a reference threshold. A probabilistic-bit neuron also has two outputs, but its input controls their probabilities. All three have discontinuous realized outputs, so ordinary derivatives of the observed samples do not provide the useful learning signal supplied by smooth artificial neurons.

Surrogate-gradient training assigns a chosen smooth derivative to the spiking threshold (Neftci et al., 2019). Straight-through estimators provide a related construction for deterministic discrete units (Bengio et al., 2013). For probabilistic bits, the derivative of an assumed mean response supplies another surrogate. Our experiments test a common measurement-based rule across these response types. They compare with the appropriate surrogate for each setting, rather than treating a single surrogate as the definition of BP for every non-differentiable network.

# B Design principles of Perturbation Propagation

## B.1 Four design principles

We developed PerProp around four design principles motivated by biological plausibility, direct learning on physical neural networks, and scalability.

**Measure neuron responses directly.** The brain learns through its neurons and synapses without a digital twin of their dynamics. Physical neurons differ from one another and drift with temperature and age, creating mismatches between their responses and a digital model. In physics-aware training, such mismatches can bias the estimated gradient (Wright et al., 2022). Systematic update bias can produce limit cycles and error floors in physical learning networks (Dillavou et al., 2025). We therefore measure neuron sensitivities and the forward responses used to learn feedback directly on the network being trained.

**Use local perturbations in one network.** Equilibrium propagation and coupled learning compare free and nudged network states (Scellier & Bengio, 2017; Stern et al., 2021). Obtaining these states involves network relaxation, and some physical implementations maintain two matched network copies to measure them simultaneously (Dillavou et al., 2025). Finite nudges also introduce gradient-estimation bias whose reduction enabled equilibrium propagation to train deep networks (Laborieux et al., 2021). We therefore measure perturbation responses locally within each layer of a single network, without rerunning the rest of the network for each measurement.

**Keep the measurement cost scalable.** Recent zeroth-order methods improve efficiency through sparse or low-rank perturbations, reducing the dimension of the perturbation subspace (Liu et al., 2025; Chen et al., 2025). We measure individual neuron responses in parallel and learn reusable feedback matrices from local layer responses. For a fixed number of probes, feedback learning adds computation proportional to the number of weights, shared across the training batch. Our experiments demonstrate that PerProp benefits from increasing model size and training resources (Section 3.1).

**Learn feedback for backward credit assignment.** Feedback matrices learned from local activity can approximate the transposed forward weights used in backpropagation, without requiring synaptic strengths to be copied between pathways (Kolen & Pollack, 1994; Akrout et al., 2019). We therefore combine measured neuron sensitivities with locally learned feedback to propagate the output error backward, so that each layer's update is guided by the same loss.

## B.2 Properties

These principles lead to the following properties of PerProp.

**Scalability.** PerProp benefits from larger models and greater training resources, achieving scaling rates comparable to BP as model size and training FLOPs increase, and steeper rates along the data, batch-size, and depth axes (Section 3.1). The measurement design keeps its additional work practical at scale: neuron slopes are measured elementwise in parallel, and the feedback matrices $B_\ell$ are learned once per update and reused across the batch. This avoids estimating the full parameter gradient from scalar loss perturbations, whose variance grows with parameter count (Oripov et al., 2025).

**Computational cost.** The forward and backward matrix products have the same arithmetic cost as in backpropagation. Additional work consists of two nonlinear evaluations per layer and, after each optimizer step, $m$ forward probes per linear layer followed by the feedback correlation update. For fixed $m$, this feedback overhead scales linearly with the number of weights and is amortized over the training batch.

**Gradient quality.** At a fixed neuron input, the measured slope is an unbiased estimate of the derivative of the response uniformly averaged over a window of width $2\varepsilon$, provided the readout noise has zero mean. This derivative exists almost everywhere even for discontinuous responses. For smooth networks with noiseless measurements, PerProp recovers the backpropagation update in the small-perturbation, aligned-feedback limit: an $O(\varepsilon^2)$ bias from finite-difference slope estimation and a residual from feedback mismatch, both vanishing as $\varepsilon \to 0$ and $B_\ell \to W_\ell^\top$ (Appendix D).

# C Implementation of Perturbation Propagation

This appendix specifies the measurements and update schedules used in the experiments. The common operations are measuring neuron sensitivities, learning a separate feedback map from input–response correlations, and propagating loss-derived errors through these local operators. Table 5 identifies the signals used in each setting. Numerical settings are collected in Appendix E.

## C.1 Dense layers and the training step

A linear layer with $d$ inputs and $p$ outputs has $W \in \mathbb{R}^{p\times d}$ and feedback $B \in \mathbb{R}^{d\times p}$. For independent Gaussian probes $\xi_r \in \mathbb{R}^d$ and measured responses $q_r = W\xi_r$, form

$$C_{ji} = \frac{\sum_{r=1}^{m}(\xi_r)_j(q_r)_i}{\sum_{r=1}^{m}(\xi_r)_j^2}, \qquad B \leftarrow (1-\rho)B + \rho C. \tag{7}$$

This update estimates feedback entries from forward responses. It does not scan input basis vectors or assign $B = W^\top$. These probes are separate from the paired neuron evaluations used to measure slopes.

Feedback entries are independently initialized from a zero-mean Gaussian with the empirical standard deviation of the corresponding forward weights. Twenty feedback updates provide an initial estimate. After each optimizer step, a new probe batch measures the updated forward map and refreshes $B$ for the next minibatch. Algorithm 1 gives the schedule for the layered network in Section 2.1.

**Algorithm 1** PerProp for linear maps and elementwise neurons

**Require:** Weights $\{W_\ell\}$, perturbation size $\varepsilon$, probe count $m$, feedback rate $\rho$, optimizer
1: Independently initialize each $B_\ell$ in the feedback orientation.
2: **for** $k = 1, \dots, 20$ **do**
3: Draw fresh probes, measure each $W_\ell\xi_{\ell,r}$, and update $B_\ell$ by (7).
4: **end for**
5: **for** each training minibatch **do**
6: Compute and retain $z_\ell = W_\ell a_{\ell-1}$ and $a_\ell = \phi_\ell(z_\ell)$.
7: Measure $s_\ell = [\phi_\ell(z_\ell + \varepsilon\mathbf{1}) - \phi_\ell(z_\ell - \varepsilon\mathbf{1})]/(2\varepsilon)$.
8: Evaluate each example's loss $\mathcal{L}_n$ and output error $v_L = \nabla_{a_L}\mathcal{L}_n$.
9: **for** $\ell = L, L-1, \dots, 1$ **do**
10: Form $g_\ell = s_\ell \odot v_\ell$ and $\widehat{G}_\ell = \langle g_\ell a_{\ell-1}^\top \rangle$.
11: **if** $\ell > 1$ **then**
12: Pass the error to the preceding layer: $v_{\ell-1} = B_\ell g_\ell$.
13: **end if**
14: **end for**
15: Apply the optimizer to $\{W_\ell\}$ using $\{\widehat{G}_\ell\}$.
16: Measure the updated linear maps with fresh probes and refresh every $B_\ell$.
17: **end for**

The backward quantities in the algorithm are per example; $\langle\cdot\rangle$ averages their outer products with the normalization of the minibatch loss. For token losses, the same reduction also averages over the contributing positions. Bias updates reduce the local errors with this normalization; feedback probes exclude biases. A linear output layer has slope one. The local weight-update estimates are supplied to SGD, momentum SGD, or AdamW as specified in Appendix E.

**Storage convention.** The code stores feedback in the same tensor shape as $W$ and applies `G.matmul(B_code)` to row-batched errors $G$. Consequently $B = B_{\text{code}}^\top$ in the paper's column-vector notation. This is a storage convention: the backward operation reads the learned tensor, without accessing the transpose of the forward weights. Hybrid instead applies $W^\top$ to the local error while retaining the same measured neuron slopes.

## C.2 Convolutions and structured operations

**Convolutions.** Let $X_{r,t,j}$ denote coordinate $j$ of the input patch at output position $t$ in probe map $r$, and $Y_{r,t,i}$ its output channel $i$. Within each convolution group, the feedback-kernel target is

$$C_{ij} = \frac{\sum_{r,t} Y_{r,t,i}X_{r,t,j}}{\sum_{r,t} X_{r,t,j}^2}. \tag{8}$$

For a two-dimensional kernel of size $k$, the Gaussian probe maps have side length $\max(8, 2k)$; one-dimensional probes have length $\max(16, 4k)$. The implementation uses 96 maps, preserves

Table 5: Feedback identification in the reported experiments. Every PerProp row learns feedback from measured responses.

| Setting | Measurements | Feedback statistic and schedule |
|---|---|---|
| Dense ANN, scaling, binary units | 96 Gaussian input vectors and linear outputs | Normalize each probe batch, then EMA at 0.1; 20 initial updates and one after each weight update. |
| Convolutions | 96 Gaussian feature maps and convolutional outputs | Correlate input patches with outputs, normalize per kernel coordinate, then the same EMA schedule. |
| Crossbar simulation | 96 converted Gaussian inputs and noisy array outputs | Batch-normalized correlation, then EMA, quantization, and feedback programming. |
| Spiking simulations | Paired-rollout spike differences and synaptic-current differences | Normalize over examples and time; update once per minibatch before the weight step. |
| BrainScaleS-2 | Hidden spike-count differences and downstream analog current-response differences | Maintain running correlation and power on the PPU; take their ratio after each training example. |

stride, padding, dilation, and groups, and clamps denominators below at $10^{-12}$. Overlapping patches share random entries, so their positions cannot be counted as independent Gaussian samples in Proposition 1.

The learned kernel specifies the convolutional input-error operator, including its spatial indexing. Applying this operator differs from transposing a compact kernel tensor. Weight updates correlate output errors with stored input patches. No input error is needed for the external data, although the reported implementations still maintain the first layer's feedback state.

**Residual paths, normalization, and embeddings.** At a residual sum, both branches receive the error, and the identity branch passes it unchanged. Reshapes, activation-array transpositions, sums, and bilinear products use their known arithmetic rules. Embedding updates accumulate errors at the selected vocabulary indices. LayerNorm, RMSNorm, and ConvNeXt's global response normalization retain their implemented arithmetic differentiation. The digital experiments thus replace feedback through learned affine maps and sensitivity through the designated neuron interfaces, while the surrounding structural computations remain explicit.

**Transformer attention.** For scores $U = QK^\top/\sqrt{d_h}$, the implementation applies the causal mask, centers each row, evaluates scalar exponential responses, divides by the row sum, and multiplies by $V$. PerProp and Hybrid measure the exponential slopes using paired scalar evaluations. Query, key, value, and output projections use learned feedback in PerProp and forward-weight transposes in Hybrid. Score products, row normalization, and value mixing use their explicit arithmetic rules. The fused implementation evaluates the same perturbed neuron responses with fewer intermediate tensors and kernel launches. The FLOP estimate in Appendix F uses a common full-attention counting convention for all arms.

**Mamba-2.** The implementation uses the quadratic state-space dual computation at context length 128. It measures scalar SiLU responses after the causal convolution and measures the two-input gate $(y, z) \mapsto y\ \mathrm{SiLU}(z)$ using four coordinate-perturbed evaluations. Trainable linear and convolutional maps use learned feedback. RMS normalization, state-space mixing, and its time-step and decay parameterizations retain their explicit computational rules. The state-space coefficient arrays conventionally named $B$ and $C$ in Mamba are distinct from the learned feedback matrices $B_\ell$. In the language-model experiments, LayerNorm and RMSNorm, attention score and value products, row normalization, residual additions, and Mamba-2's softplus and state-space computation use analytical derivatives. These operations form the digital computation surrounding the measured neuron responses and learned synaptic feedback.

### C.3 Spiking neurons and three-factor updates

**Neuron-state measurements.** Write a spiking neuron's transition as $F(I_t, x_{t-1}) = (s_t, x_t)$, where $I_t$ is its input current, $s_t$ its emitted spike, and $x_t$ its membrane and any adaptation state. PerProp perturbs each local input coordinate and contracts the response differences with the error arriving at $(s_t, x_t)$. For coordinate $j$ and output error $v$,

$$\widehat{[J_F^\top v]}_j = \frac{\langle v, F(u + \varepsilon e_j) - F(u - \varepsilon e_j)\rangle}{2\varepsilon}, \qquad u = (I_t, x_{t-1}). \tag{9}$$

Neurons are evaluated in parallel. LIF, QIF, and EIF transitions have two local inputs and require four additional evaluations; the Izhikevich-form and ALIF transitions have three and require six. Temporal error propagation uses these measured transitions and learned synaptic feedback. The simulations retain the unrolled states needed for this computation.

**Feedback from spikes.** Separate nominal, positive, and negative rollouts start from the same zero state. Independent random signs perturb currents across examples, neurons, and time. The perturbed rollouts evolve their own local states; in recurrent networks, both receive the recurrent drive from the nominal trajectory. For an outgoing synaptic map $W$, collect spike differences $x_r = (s_r^+ - s_r^-)/(2\varepsilon)$ and current responses $y_r = Wx_r$. Use (7) with $x_r, y_r$ in place of $\xi_r, q_r$, pooling examples and time. Coordinates whose mean squared spike difference is at most $10^{-6}$ retain their previous feedback. This refresh occurs after error propagation and before the forward-weight optimizer step. Its rollouts are separate from the coordinate measurements in (9).

**Conditional unbiasedness of the target.** Condition on the nominal trajectory and forward weights. Reversing a neuron's entire perturbation-sign sequence exchanges its positive and negative trajectories, negating its spike-difference sequence. Independent signs and the common nominal recurrent drive make cross-neuron correlations vanish in expectation. For an active coordinate $j$,

$$C_{ji} = W_{ij} + \sum_{k \neq j} W_{ik} \frac{\sum_r x_{r,j} x_{r,k}}{\sum_r x_{r,j}^2}.$$

Conditional on neuron $j$'s sequence, each cross term has zero mean by the sign symmetry for neuron $k$. The target is therefore conditionally unbiased. The active-coordinate decision depends only on that coordinate's squared responses and preserves the symmetry. Inactive coordinates receive no new estimate. These spike differences need not be Gaussian or independent over time; the Gaussian variance formula does not apply to them.

**The three factors.** The current error at a synapse is the received error multiplied by the measured postsynaptic sensitivity. Multiplying it by presynaptic activity gives

$$\Delta W_{ij} = -\eta \sum_t a_{j,t}\, \chi_{i,t}\, e_{i,t}, \tag{10}$$

where $a_{j,t}$ is presynaptic activity, $\chi_{i,t}$ is measured postsynaptic sensitivity, and $e_{i,t}$ is the loss-derived error at that response. With several state outputs, the contraction in (9) sums their contributions to the current error. This identifies the signals needed at a synapse without imposing a surrogate spike derivative. Appendix I maps these factors to spike counts, analog current measurements, and PPU arithmetic in the finite-window hardware experiment.

# D MATHEMATICAL ANALYSIS

We establish what the local measurements estimate and when the resulting update approaches backpropagation. The neuron result concerns a single response. The feedback results concern a fixed linear map and independent probe batches. These support the gradient-quality property in Section B.2; the scaling results in Section 3.1 are empirical.

## D.1 MEASURED NEURON SLOPES

For a locally integrable scalar response $\phi$, define its uniform local average

$$\bar{\phi}_\varepsilon(z) = \frac{1}{2\varepsilon}\int_{z-\varepsilon}^{z+\varepsilon} \phi(u)\,du. \tag{11}$$

It is locally absolutely continuous. The fundamental theorem for Lebesgue integrals gives, almost everywhere,

$$\bar{\phi}'_\varepsilon(z) = \frac{\phi(z+\varepsilon) - \phi(z-\varepsilon)}{2\varepsilon}. \tag{12}$$

Paired forward measurements therefore recover the derivative of this local average even when $\phi$ has jumps. In an elementwise layer, two vector evaluations supply all scalar differences simultaneously.

For a stochastic measurement $Y(z)$ with finite mean $\mu(z) = \mathbb{E}Y(z)$, assume each member of the pair has the correct marginal distribution at its perturbed input. Then

$$\mathbb{E}\frac{Y(z+\varepsilon) - Y(z-\varepsilon)}{2\varepsilon} = \frac{\mu(z+\varepsilon) - \mu(z-\varepsilon)}{2\varepsilon} = \bar{\mu}'_\varepsilon(z)$$

almost everywhere. The pair need not be independent; shared randomness can reduce its difference variance. Independent zero-mean readout noise with variance $\sigma^2$ in each measurement contributes slope variance $\sigma^2/(2\varepsilon^2)$. Averaging $n$ independent pairs divides it by $n$. This is measurement noise, separate from random-direction projection error.

If $\phi$ is three times differentiable on the measurement interval and $|\phi'''| \le M_3$, Taylor's theorem yields

$$\left|\frac{\phi(z+\varepsilon) - \phi(z-\varepsilon)}{2\varepsilon} - \phi'(z)\right| \le \frac{M_3}{6}\varepsilon^2. \tag{13}$$

Equation (12) is a local identity. The nominal forward network still evaluates $\phi$, rather than $\bar{\phi}_\varepsilon$, so it does not identify the complete finite-$\varepsilon$ update as the exact gradient of a globally smoothed network.

## D.2 LEARNING A FIXED FEEDBACK MAP

Fix $W \in \mathbb{R}^{p\times d}$. Draw $m > 2$ independent probes $\xi_r \sim \mathcal{N}(0, I_d)$, measure $q_r = W\xi_r$ without noise, and form $C$ by (7). Each update uses an independent probe batch. This subsection uses the ideal mathematical normalization without clipping or quantization.

**Proposition 1** (Normalized-correlation target)**.** *The target is unbiased, $\mathbb{E}C = W^\top$, and*

$$V := \mathbb{E}\|C - W^\top\|_F^2 = \frac{d-1}{m-2}\|W\|_F^2. \tag{14}$$

*For fixed $u \in \mathbb{R}^p$, or conditionally for a random $u$ independent of the probes,*

$$\mathbb{E}\|(C - W^\top)u\|^2 = \frac{d-1}{m-2}\|W^\top u\|^2. \tag{15}$$

*Proof.* Let $x_j = ((\xi_1)_j, \dots, (\xi_m)_j)^\top$. Expansion gives

$$C_{ji} - W_{ij} = \sum_{k\ne j} W_{ik}\frac{x_j^\top x_k}{\|x_j\|^2}.$$

Conditional on $x_j$, the other $x_k$ are independent standard Gaussian vectors, so each summand has zero mean and

$$\mathbb{E}[(C_{ji} - W_{ij})^2 \mid x_j] = \frac{\sum_{k\ne j} W_{ik}^2}{\|x_j\|^2}.$$

Since $\|x_j\|^2 \sim \chi^2_m$ and $\mathbb{E}[1/\|x_j\|^2] = 1/(m-2)$, expectation and summation over $i, j$ give (14). For (15), set $h = W^\top u$ and apply the same calculation to $[(C - W^\top)u]_j = \sum_{k\ne j} h_k(x_j^\top x_k)/\|x_j\|^2$. □

**Proposition 2** (Constant-rate feedback learning)**.** *Let $B_t = (1-\rho)B_{t-1} + \rho C_t$, with $0 < \rho \le 1$ and $B_0$ independent of future probes. If $R_0 = \mathbb{E}\|B_0 - W^\top\|_F^2 < \infty$, then*

$$\mathbb{E}B_t - W^\top = (1-\rho)^t(\mathbb{E}B_0 - W^\top), \tag{16}$$

$$\mathbb{E}\|B_t - W^\top\|_F^2 = (1-\rho)^{2t}R_0 + \frac{\rho}{2-\rho}[1-(1-\rho)^{2t}]V. \tag{17}$$

*Proof.* Set $E_t = B_t - W^\top$ and $N_t = C_t - W^\top$. Then $E_t = (1-\rho)E_{t-1} + \rho N_t$. The innovation $N_t$ has zero mean, second moment $V$, and is independent of $E_{t-1}$. Expectation gives (16). Squaring eliminates the expected cross term, giving $R_t = (1-\rho)^2 R_{t-1} + \rho^2 V$. Summing this geometric recurrence proves (17). □

The mean approaches $W^\top$, while constant-rate learning retains the variance floor

$$\lim_{t\to\infty} \mathbb{E}\|B_t - W^\top\|_F^2 = \frac{\rho}{2-\rho}\frac{d-1}{m-2}\|W\|_F^2. \tag{18}$$

For $m = 96$ and $\rho = 0.1$, the normalized floor is $(d-1)/1786$. Constant-rate feedback learning therefore remains noisy even after many updates.

**Proposition 3** (Decreasing-rate convergence)**.** *For the same fixed map and independent targets, choose $\rho_t = 1/t$ for $t \ge 1$. Then $B_t = t^{-1}\sum_{k=1}^t C_k$ and*

$$\mathbb{E}B_t = W^\top, \qquad \mathbb{E}\|B_t - W^\top\|_F^2 = V/t. \tag{19}$$

*Thus $B_t \to W^\top$ in mean square and almost surely.*

*Proof.* The averaging identity follows by induction, with the first update having rate one. Independent zero-mean innovations give variance $tV/t^2$. Each target entry has finite second moment, so the strong law applies to the finitely many entries and yields almost-sure matrix convergence. □

The decreasing-rate schedule establishes convergence for a fixed map; the experiments use a constant rate. When $C_t$ instead measures changing weights $W_t$, the exact error recurrence is

$$B_t - W_t^\top = (1-\rho)(B_{t-1} - W_{t-1}^\top) - (1-\rho)(W_t - W_{t-1})^\top + \rho(C_t - W_t^\top). \tag{20}$$

It separates retained estimation error, weight movement, and new measurement noise. Historical feedback influences current weights, so the fixed-map results do not assert exact conditional unbiasedness of the complete training update.

### D.3 Consistency in a fixed smooth network

Fix a finite network, its nominal forward states, and its output loss derivative. Define $D_\ell = \mathrm{diag}(\phi_\ell'(z_\ell))$, $S_\ell = \mathrm{diag}(s_\ell)$, and

$$A_\ell = W_\ell^\top D_\ell, \qquad \widehat{A}_\ell = B_\ell S_\ell.$$

The local operator difference satisfies

$$\|\widehat{A}_\ell - A_\ell\|_2 \le \|B_\ell - W_\ell^\top\|_2\|S_\ell\|_2 + \|W_\ell\|_2\|S_\ell - D_\ell\|_2. \tag{21}$$

For any finite product, telescoping gives

$$\widehat{A}_1\cdots\widehat{A}_n - A_1\cdots A_n = \sum_{k=1}^n A_1\cdots A_{k-1}(\widehat{A}_k - A_k)\widehat{A}_{k+1}\cdots\widehat{A}_n. \tag{22}$$

Taking norms bounds each propagated error by a sum of local errors multiplied by finite prefix and suffix norms. Applying this identity to each backward suffix, then including the fixed presynaptic activations and local slopes, yields

$$\|\widehat{G} - G^{\mathrm{BP}}\| \le C_\varepsilon\varepsilon^2 + \sum_\ell C_\ell\|B_\ell - W_\ell^\top\|_2 \tag{23}$$

in a bounded neighborhood of the aligned-feedback limit. Here $\widehat{G}$ concatenates the weight updates, and the constants depend on the fixed network, its states, and its propagation norms. Equation (13) supplies the $O(\varepsilon^2)$ term. Therefore noiseless measurements recover BP as $\varepsilon \to 0$ and $B_\ell \to W_\ell^\top$. This finite-network result does not prescribe convergence of the coupled optimizer or a loss-scaling exponent.

# E EXPERIMENTAL SETTINGS

This appendix specifies the architectures, training settings, and evaluation protocols. Unless stated otherwise, repeated experiments use seeds 0, 1, and 2, and report the mean and sample standard deviation. The scaling experiments use seed 0 for each configuration. GPT with OPZO has two completed seeds; its third seed diverged.

For the non-spiking PerProp experiments, feedback learning uses 96 independent Gaussian probes per update, normalized input–response correlations, an exponential moving average with rate $\rho = 0.1$, 20 feedback updates before training, and one feedback update after every optimizer step. Neuron-slope measurements and feedback probes are evaluated separately. The spiking simulations and BrainScaleS-2 use the spike-response and hardware implementations described below.

## E.1 SCALING EXPERIMENTS

**Data and architecture.** All scaling runs use the same cached FineWeb-Edu[1] token streams: 6,200,006,482 training tokens and 5,052,597 validation tokens, encoded with the GPT-2 tokenizer and its 50,257-token vocabulary. Training samples contiguous windows of 1,024 tokens at uniformly drawn starting positions, with the next token as the prediction target. The models are causal Transformers with learned positional embeddings, untied input embeddings and output weights, pre-RMSNorm with numerical constant $10^{-5}$, and SwiGLU feed-forward layers. For embedding width $d$, the number of attention heads is $d/64$, and the feed-forward hidden width is $8d/3$ rounded to the nearest multiple of 64. Residual output projections are initialized with a factor $1/\sqrt{2L}$ for $L$ Transformer layers.

**Optimization and budgets.** We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay 0.1, and global gradient-norm clipping at 1. The learning rate increases linearly during warmup and then follows cosine decay to 10% of its peak. PerProp and Hybrid use neuron perturbation size $\varepsilon = 0.01$. Table 6 gives the executed configurations. Its parameter column counts all trainable parameters, including embeddings and the output head. The token-budget convention uses the non-embedding approximation $N_{\text{budget}} = 12Ld^2$; the model-size, depth, and additional width runs allocate approximately $20N_{\text{budget}}$ training tokens.

Table 6: Scaling configurations. Batch size is measured in tokens per optimizer update; parameter counts are in millions. Rows shared by several sweeps appear once. The first 17 rows supply the main figure; the final three are additional width configurations.

| Sweep | $L$ | $d$ | Parameters | Batch tokens | Updates | Warmup | Peak LR |
|---|---|---|---|---|---|---|---|
| Model size | 3 | 384 | 44.302 | 24,576 | 4,320 | 216 | $1.2\times10^{-3}$ |
| Model size / reference | 6 | 384 | 49.612 | 32,768 | 6,480 | 324 | $1.2\times10^{-3}$ |
| Model size | 12 | 384 | 60.234 | 49,152 | 8,640 | 432 | $9\times10^{-4}$ |
| Model size (PerProp) | 12 | 768 | 162.935 | 98,304 | 17,280 | 863 | $6\times10^{-4}$ |
| Model size | 24 | 1,024 | 407.588 | 172,032 | 35,108 | 1,755 | $3\times10^{-4}$ |
| Data: $2.5N_{\text{budget}}$ tokens | 6 | 384 | 49.612 | 32,768 | 810 | 100 | $1.2\times10^{-3}$ |
| Data: $5N_{\text{budget}}$ tokens | 6 | 384 | 49.612 | 32,768 | 1,620 | 100 | $1.2\times10^{-3}$ |
| Data: $10N_{\text{budget}}$ tokens | 6 | 384 | 49.612 | 32,768 | 3,240 | 162 | $1.2\times10^{-3}$ |
| Data: $40N_{\text{budget}}$ tokens | 6 | 384 | 49.612 | 32,768 | 12,960 | 648 | $1.2\times10^{-3}$ |
| Data: $80N_{\text{budget}}$ tokens | 6 | 384 | 49.612 | 32,768 | 25,920 | 1,296 | $1.2\times10^{-3}$ |
| Depth | 3 | 384 | 44.302 | 32,768 | 3,240 | 162 | $1.2\times10^{-3}$ |
| Depth | 12 | 384 | 60.234 | 32,768 | 12,960 | 648 | $9\times10^{-4}$ |
| Depth | 24 | 384 | 81.477 | 32,768 | 25,920 | 1,296 | $6\times10^{-4}$ |
| Batch size | 6 | 384 | 49.612 | 8,192 | 6,480 | 324 | $1.2\times10^{-3}$ |
| Batch size | 6 | 384 | 49.612 | 16,384 | 6,480 | 324 | $1.2\times10^{-3}$ |
| Batch size | 6 | 384 | 49.612 | 65,536 | 6,480 | 324 | $1.2\times10^{-3}$ |
| Batch size | 6 | 384 | 49.612 | 131,072 | 6,480 | 324 | $1.2\times10^{-3}$ |
| Additional width | 6 | 192 | 22.152 | 32,768 | 1,620 | 100 | $1.2\times10^{-3}$ |
| Additional width | 6 | 768 | 120.458 | 32,768 | 25,920 | 1,296 | $6\times10^{-4}$ |
| Additional width | 6 | 1,536 | 325.852 | 32,768 | 103,680 | 5,184 | $3\times10^{-4}$ |

The six-layer, width-384 reference supplies the 20-token data point, the six-layer depth point, and the 32,768-token batch point. The data sweep holds the architecture and batch size fixed while

[1] https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu

changing the number of updates. The batch sweep holds the architecture and update count fixed, so larger batches process more tokens. The depth sweep keeps width and batch size fixed, increases the number of updates proportionally with depth, and reduces the peak learning rate for the deeper models. The parameter, FLOP, and energy panels use the same five model-size configurations.

Architectures and total token budgets are matched across BP, Hybrid, and PerProp. For the 162.935M-parameter model, BP and Hybrid use 92,160 tokens per batch, 18,432 updates, and 921 warmup updates; PerProp uses the corresponding row in Table 6. Both schedules process 1,698,693,120 tokens. The largest model processes 6,039,699,456 tokens in each arm. Gradient accumulation and distributed training implement the global batch sizes in the table.

**Evaluation and execution.** Validation runs every 200 updates and at the final update. Each evaluation averages negative log-likelihood over 20 randomly sampled minibatches, using the training microbatch size in Table 7; the additional width-768 PerProp run uses 40 minibatches. The plotted score averages the logged rank-0 validation losses over the final 10% of updates, or the final 500 updates when this is longer, and then exponentiates that average to obtain perplexity. Scores use the fixed training budget and final evaluation window. The runs use FP32 model tensors with TF32 matrix multiplication enabled. PerProp uses fused neuron evaluations; the width-192 run and the three-layer depth run switch to this implementation after update 1,000. See Appendix F for the scaling results and FLOP accounting, and Appendix K.4 for projected energy.

Table 7: Scaling microbatch and parallelism settings. Entries give sequences per GPU $\times$ GPU count. Gradient accumulation supplies the global token batch in Table 6.

| Configuration | BP / Hybrid | PerProp |
|---|---|---|
| Default | $16 \times 1$ | $16 \times 1$ |
| Smallest model-size run ($L = 3, d = 384$) | $12 \times 1$ | $12 \times 1$ |
| 8,192-token batch run | $8 \times 1$ | $8 \times 1$ |
| 162.935M-parameter model | $15 \times 2$ | $24 \times 4$ |
| 407.588M-parameter model | $3 \times 4$ | $7 \times 8$ |
| Additional width 768 | $16 \times 1$ | $8 \times 4$ |
| Additional width 1,536 | $8 \times 4$ | $8 \times 4$ |

### E.2 Standard architectures

**Architectures.** Table 8 summarizes the networks used to compare learning rules. GPT uses pre-LayerNorm blocks, causal attention, GELU feed-forward layers of width 1,024, learned positional embeddings, and untied input and output weights. Mamba-2 uses pre-RMSNorm blocks, expansion factor 2, state dimension 64, head dimension 64, a width-4 depthwise causal convolution, and one group of input-dependent state-space parameters. Its 512-dimensional expanded representation has eight heads. The implementation evaluates the state-space dual computation in its quadratic form at context length 128.

ResMLP-16 projects flattened CIFAR-10 images to width 512 and applies 16 residual blocks. Each branch consists of LayerNorm, a $512\times512$ linear map, GELU, and a second $512\times512$ map, with the latter's weights scaled by $1/\sqrt{32}$ at initialization. A final LayerNorm and linear classifier produce ten logits. ConvNeXt V2 has three stages with depths $(3, 3, 3)$ and widths $(64, 128, 256)$. It uses a $2\times2$ stride-2 stem and downsampling layers, $7\times7$ depthwise convolutions, fourfold pointwise expansion, GELU, global response normalization, and global average pooling. Forward–forward uses its label-conditioned, 16-layer ReLU MLP of width 512 in the ResMLP comparison; this model has 5,533,706 parameters.

Table 8: Architectures and training budgets for learning-rule comparisons. TinyStories evaluates the same language-model checkpoints trained on WikiText-103.

| Architecture | Parameters | Batch | Context | Training budget |
|---|---|---|---|---|
| GPT, 24 layers, width 256 | 23,181,824 | 64 | 128 | 3,000 updates |
| Mamba-2, 24 layers, width 256 | 14,563,136 | 64 | 128 | 3,000 updates |
| ResMLP-16, width 512 | 10,000,906 | 128 | image | 30 epochs |
| ConvNeXt V2, $(3, 3, 3)$ blocks | 2,320,394 | 128 | image | 30 epochs |

**WikiText-103 training and evaluation.** We train an 8,192-token byte-level BPE tokenizer on the raw WikiText-103 training split (Merity et al., 2016), with minimum token frequency 2 and an end-of-text special token. The same tokenizer encodes the original training, validation, and test splits. Training draws contiguous 128-token windows uniformly from the training stream. Each update

processes 8,192 tokens, for 24.576 million tokens over 3,000 updates. Optimization uses SGD with momentum $0.9$, zero weight decay, and FP32 model tensors with matrix TF32 disabled. For update index $t$, peak learning rate $\eta_{\max}$, and budget $S = 3{,}000$, the schedule is

$$\eta_t = \eta_{\max} \min\left(1, \frac{t+1}{100}\right) \frac{1 + \cos(\pi \min(t, S)/S)}{2}.$$

Validation is evaluated every 500 updates using 20 batches of 64 sequences. Table 1 reports the final validation NLL in nats. PerProp and Hybrid use $\varepsilon = 0.01$ for GPT and $\varepsilon = 0.1$ for Mamba-2.

Table 9: Selected peak learning rates for the language-model comparisons. All entries use SGD with momentum $0.9$.

| Learning rule | GPT | Mamba-2 |
|---|---|---|
| BP | $0.2$ | $0.1$ |
| Hybrid | $0.2$ | $0.3$ |
| PerProp | $1/15$ | $0.1$ |
| Feedback alignment | $0.1$ | $0.1$ |
| Direct feedback alignment | $0.6$ | $0.1$ |
| DRTP | $0.03$ | $0.03$ |
| Weight perturbation / MeZO | $3\times10^{-4}$ | $3\times10^{-4}$ |
| OPZO | $0.03$ | $0.01$ |

The completed GPT PEPITA and node-perturbation runs use learning rates $1/3$ and $0.03$, respectively. Feedback alignment clips the global gradient norm at 1; the other reported language-model configurations use no gradient clipping. Direct-feedback scales are $0.1$ for DFA and $10^{-6}$ for DRTP. GPT uses full layerwise DFA: independent fixed feedback projections supply errors to the hidden affine responses, embeddings, and normalization sites. The 194 feedback matrices have Gaussian entries with variance $1/8{,}192$ before applying the feedback scale and are shared across token positions.

**Language-model hyperparameter selection.** For each replacement configuration other than weight perturbation, seed-0 pilots compare the previously selected learning rate with one third of that rate over 200 updates, using the full 3,000-update schedule. Feedback alignment also tries the previous rate with clipping at norm 1. Full GPT DFA is selected from learning rates $\{1/15, 0.2, 0.6\}$ and feedback scales $\{0.1, 1, 3\}$ using the same pilot budget. Validation NLL selects a setting for all three full runs. Mamba-2 BP uses learning rate $0.1$ for all seeds after the $0.3$ setting diverged in a full run. Pilot scores are excluded from the reported results.

**TinyStories evaluation.** We evaluate the final WikiText-103 checkpoints on all 21,990 stories in the TinyStories validation split (Eldan & Li, 2023), using the existing WikiText tokenizer and no additional training. After removing outer whitespace from each story, we prepend one unscored end-of-text token. Evaluation uses context length 128 and stride 64: the first window scores up to 128 text tokens, and subsequent windows score only the newly introduced tokens. Each text token is scored once, yielding 5,592,426 scored tokens. Context and Mamba state are reset at every window, and no terminal end-of-text token is included in the score. The reported NLL is the total negative log-likelihood divided by the number of scored text tokens. Models run in FP32 with matrix TF32 disabled, and loss totals accumulate in FP64.

**CIFAR-10 data and optimization.** We reserve 5,000 examples from the original 50,000-image training set using a fixed random permutation with seed 20260905, leaving 45,000 training images and the original 10,000-image test set. Channels are normalized by means $(0.4914, 0.4822, 0.4465)$ and standard deviations $(0.2470, 0.2435, 0.2616)$. ResMLP uses flattened images without augmentation. ConvNeXt uses random $32\times32$ crops after four-pixel zero padding and horizontal flips with probability $0.5$. Training lasts 30 epochs at batch size 128 with cosine learning-rate decay; final-epoch test accuracy is reported. PerProp and Hybrid use $\varepsilon = 0.01$.

DFA uses feedback scale $0.1$ on ResMLP and $10^{-3}$ on ConvNeXt. The completed ConvNeXt DRTP runs use learning rate $5\times10^{-4}$ and feedback scale $10^{-5}$. Difference target propagation uses target step $0.5$ and reconstruction-noise scale $0.1$; forward–forward uses goodness threshold 2.

**Perturbation-baseline parameters.** Weight perturbation uses displacement $10^{-3}$ and 64 Rademacher directions per update, except for GPT, which uses 16. Node perturbation and the INP-like comparison use 64 Rademacher directions in the oracle-replay implementation described in Appendix G. PEPITA uses input-error projection scale $0.05$. OPZO uses input-noise scale $0.1$ and correlation momentum $0.99$, except on ConvNeXt, where the momentum is $0.9$. PseuZO and

Table 10: Selected CIFAR-10 optimization settings. ResMLP uses zero weight decay; ConvNeXt uses AdamW with weight decay $0.05$. SGD uses momentum $0.9$; Adam and AdamW use $(\beta_1, \beta_2) = (0.9, 0.999)$.

| Learning rule | ResMLP optimizer | ResMLP LR | ConvNeXt LR |
|---|---|---|---|
| BP / Hybrid / PerProp | SGD | $0.03$ | $0.002$ |
| Feedback alignment | Adam | $10^{-4}$ | $0.002$ |
| Direct feedback alignment | SGD | $0.01$ | $5\times10^{-4}$ |
| PEPITA | SGD | $0.01$ | $2\times10^{-4}$ |
| Difference target propagation | SGD | $0.01$ | n/a |
| Forward–forward | Adam | $0.001$ | n/a |
| Weight perturbation / MeZO | SGD | $3\times10^{-5}$ | $2\times10^{-4}$ |
| Node perturbation | SGD | $3\times10^{-4}$ | $5\times10^{-4}$ |
| INP | SGD | $0.003$ | n/a |
| OPZO | SGD | $0.003$ | $5\times10^{-4}$ |
| PseuZO / PseuZO-NP | SGD | $0.03$ | $0.002$ |

PseuZO-NP use displacement $0.01$, a history window of 16, one probe, and momentum $0.9$. The additional ResMLP BOND and HZO runs use SGD at learning rate $0.03$, displacement $0.01$, and query chunks of eight; BOND uses its reservoir partition with $b = 1.19\times10^{-7}$ and moment rates $(0.9, 0.999)$, and HZO evaluates spatially independent queries in parallel.

**Timing protocol.** The Pareto figures pair the completed training scores with separately measured step times on one NVIDIA RTX A6000. Benchmarks run sequentially with two CPU threads, FP32, and TF32 disabled. After 20 warmup updates, we time three blocks of ten updates with CUDA synchronization and report the median block-average step time. Each step includes the forward pass, loss, learning-rule computation, optimizer update, and any feedback refresh. Fixed real batches are preloaded, so sampling, augmentation, evaluation, checkpoint writing, and initialization are outside this timing interval.

### E.3 Spiking neural networks

**Data and networks.** MNIST uses all 60,000 training and 10,000 test images for the final runs. Pixel intensities in $[0, 1]$ are encoded as independent Bernoulli spikes at each of 25 time steps. The network has 784 inputs, two feed-forward hidden layers of 256 spiking neurons, and ten output channels. SHD (Cramer et al., 2022b) uses its original 8,156 training and 2,264 test recordings, with 700 input channels and 20 classes. Events are assigned to 100 bins over a 1.4-second interval, with indices beyond the interval assigned to the final bin; multiple events in the same channel and bin produce a single binary input. Its network has one recurrent layer of 256 ALIF neurons. Training augmentation applies circular temporal and channel shifts, drawn uniformly from the integer ranges $[-5, 5]$ and $[-10, 10]$, respectively.

Both networks use a non-spiking leaky readout. If $z_t$ is its synaptic input, its state follows $r_t = 0.9r_{t-1} + z_t$; the time average of $r_t$ supplies the logits for cross-entropy. Neuron and readout states start at zero for each example. Feed-forward hidden weights are initialized from a zero-mean Gaussian with standard deviation $2/\sqrt{d_{\text{in}}}$ for MNIST and $4/\sqrt{d_{\text{in}}}$ for SHD. Recurrent SHD weights use standard deviation $0.5/\sqrt{256}$.

Table 11: Training settings for the spiking simulations. EIF uses the validation-selected PerProp settings shown separately. All runs use Adam with $(\beta_1, \beta_2) = (0.9, 0.999)$, zero weight decay, and batch size 128.

| Setting | Epochs | Initial LR | PerProp $\varepsilon$ | PerProp $\rho$ | LR schedule |
|---|---|---|---|---|---|
| MNIST: LIF, QIF, Izhikevich | 3 | 0.001 | 1 | 0.2 | Constant |
| MNIST: EIF | 3 | 0.001 | 0.5 | 0.5 | Constant |
| SHD: recurrent ALIF | 100 | 0.002 | 1 | 0.2 | Cosine to 0.0001 |

**Neuron dynamics.** All simulated spiking neurons have threshold 1 and reset their membrane to zero after a spike. Writing $u_t$ for the pre-reset membrane, $s_t = \mathbf{1}[u_t \geq 1]$ and $v_t = u_t(1 - s_t)$. Table 12 specifies the dynamics in normalized units; $I_t$ is the incoming current and $w_t$ is an adaptation or recovery state. The Izhikevich-form model uses the same normalized voltage and reset convention as the other models.

**PerProp, Hybrid, and surrogate BPTT.** PerProp measures the neuron-state transition by paired coordinate perturbations: two coordinates for LIF, QIF, and EIF, and three for Izhikevich-form and

Table 12: Neuron transitions used in the SNN simulations. All state variables are initialized to zero.

| Neuron | Pre-reset membrane $u_t$ | Additional state |
|---|---|---|
| LIF | $0.9v_{t-1} + I_t$ | none |
| QIF | $0.9v_{t-1} + 0.15v_{t-1}^2 + I_t$ | none |
| EIF | $0.9v_{t-1} + I_t + 0.15e^{\min((v_{t-1}-0.5)/0.15,10)}$ | none |
| Izhikevich-form | $0.9v_{t-1} + 0.15v_{t-1}^2 + I_t - w_{t-1}$ | $w_t = 0.96w_{t-1} + 0.02v_t + 0.3s_t$ |
| ALIF | $0.9v_{t-1} + I_t - 1.2w_{t-1}$ | $w_t = 0.96w_{t-1} + s_t$ |

ALIF neurons. Feedback learning uses separate positive and negative current-perturbation rollouts with independent random signs across examples, neurons, and time steps. Correlations average over the minibatch and time dimensions. Feedback entries with mean squared spike-response difference at most $10^{-6}$ remain unchanged. Feedback updates occur after the credit computation and before the forward-weight optimizer step, once per minibatch. These runs use spike-response feedback learning from Section 2.3, with rates in Table 11. Hybrid uses the same coordinate measurements with $\varepsilon = 1$ and transposed forward weights.

Surrogate BPTT uses the arctangent spike derivative $k_a(x) = a/[2(1 + (\pi ax/2)^2)]$ at $a = 2$, where $x = u_t - 1$, except for the validation-selected EIF baseline. The reset and neuron-state computations remain in the temporal differentiation graph. We report final-epoch test accuracy over seeds 0–2, with evaluation batch size 500. MNIST evaluation fixes the spike-encoding random seed at 1234.

**EIF hyperparameter selection.** Both EIF sweeps use a stratified holdout of 6,000 MNIST training examples, with 600 per class and split seed 20260918. Candidates train on the remaining 54,000 examples for three epochs. The surrogate sweep first crosses learning rates $\{0.0003, 0.001, 0.003\}$ with arctangent widths $a \in \{0.25, 0.5, 1, 2, 4, 8\}$ and sigmoid, triangular, and rectangular widths $a \in \{0.5, 1, 2, 4\}$, yielding 54 configurations. The additional kernels are

$$k_a^{\text{sigmoid}}(x) = 2a\,\sigma(2ax)(1 - \sigma(2ax)),$$
$$k_a^{\text{triangle}}(x) = \frac{a}{2}\max(1 - a|x|/2, 0),$$
$$k_a^{\text{rectangle}}(x) = \frac{a}{2}\mathbf{1}[|x| < 1/a].$$

Each kernel has unit integral and peak $a/2$. A further 16 configurations multiply arctangent or rectangular kernels by gains $\{0.25, 0.5\}$ at widths $\{2, 4\}$ and learning rates $\{0.001, 0.003\}$, for 70 surrogate configurations in total.

The PerProp sweep crosses $\varepsilon \in \{0.1, 0.3, 0.5, 1, 2, 3\}$, learning rates $\{0.0003, 0.001, 0.003\}$, and feedback rates $\{0.05, 0.2, 0.5\}$, giving 54 configurations. Each sweep first screens seed 0, then repeats its top three candidates at seeds 1 and 2. The surrogate sweep also retains the best arctangent candidate; the PerProp sweep retains the original setting if it is absent from the top three. Mean final validation accuracy selects the configuration, with validation cross-entropy breaking ties. The selected surrogate is rectangular with $a = 2$, gain 1, and learning rate $0.001$; the selected PerProp setting is $\varepsilon = 0.5$, $\rho = 0.5$, and learning rate $0.001$. Final runs retrain these fixed selections on all 60,000 training examples before evaluating the test set.

### E.4 Comparator and probabilistic-bit networks

These experiments use the ResMLP-16 architecture and CIFAR-10 split from Appendix E.2, replacing every GELU response with a comparator or probabilistic bit. A comparator returns $\text{sign}(z - \theta_j)$. A probabilistic bit returns $+1$ with probability $\sigma((z - \theta_j)/T)$ and $-1$ otherwise, with temperature $T = 0.5$. Each neuron's threshold is drawn once from $\mathcal{N}(0, 0.3^2)$ and held fixed.

All three methods train for 30 epochs with batch size 128, SGD with momentum $0.9$, initial learning rate $0.03$, cosine decay, and zero weight decay. PerProp and Hybrid use $\varepsilon = 1$; PerProp uses the Gaussian feedback schedule specified at the beginning of this appendix. We report final test accuracy over seeds 0–2.

The nominal comparator surrogate is $\mathbf{1}[|z| \le 1]$. The probabilistic-bit surrogate is the derivative of the nominal mean response, $2\sigma(z/T) - 1$, namely $2\sigma(z/T)(1 - \sigma(z/T))/T$. Threshold-aware controls replace $z$ by $z - \theta_j$ in these expressions. PerProp and Hybrid evaluate the actual thresholded responses. For probabilistic bits, the nominal and paired perturbed evaluations share the same uniform random sample for each example and neuron.

### E.5 BRAINSCALES-2 EXPERIMENTS

**Task and encoding.** We use the Yin–Yang generator (Kriener et al., 2021) with outer radius $0.5$ and dot radius $0.1$. It samples a target class uniformly from the three classes and then draws a point from that region by rejection sampling. Training, validation, and test sets contain 5,000, 1,000, and 1,000 examples, generated with fixed seeds 42, 43, and 44. A point $(x, y)$ is represented by five input channels $(x, y, 1-x, 1-y, 1)$. Each channel value $a$ is encoded as $\mathrm{round}(32a)$ evenly spaced spikes within a 0.5 ms input window.

The network has five inputs, 32 hidden analog spiking neurons, and three output analog spiking neurons. All connections are excitatory. Initial input-to-hidden weights are sampled uniformly from $[12, 36]$ and hidden-to-output weights from $[4, 12]$, then rounded to integer hardware values. Both layers are trained. Predictions use the output neuron with the largest spike count.

**Hardware configuration.** Runs use the BrainScaleS-2 PyNN interface (Pehle et al., 2022) and on-chip PPU kernels in the EBRAINS-25.10 environment. Both methods retain the same supplied neuron calibration. R-STDP additionally loads the calibrated correlation sensors used by its eligibility measurement. The host schedules examples and labels, transfers checkpoints, and computes evaluation summaries; feedback learning, error propagation, and PerProp weight updates execute on the PPU. Synaptic weights are programmed as six-bit integers in $[0, 63]$, while fractional update accumulators are retained on the PPU.

Table 13: BrainScaleS-2 training settings. PerProp and R-STDP share the network, data, initial weights, example order, training epochs, and checkpoint-selection rule.

| Setting | PerProp | R-STDP |
|---|---|---|
| Training epochs / examples per epoch | 5 / 5,000 | 5 / 5,000 |
| Learning rate | 1 | 0.03 |
| Forward-weight warmup | 128 examples | 128 examples |
| Training period per example | 20 ms | 20 ms |
| Evaluation period per example | 5 ms | 5 ms |
| Examples per host execution chunk | 128 | 128 |
| Feedback correlation averaging rate | 0.02 | n/a |
| Reward-baseline averaging rate | n/a | 0.05 |
| Exploration weight amplitude | n/a | 8 |
| Homeostatic coefficient | n/a | 1 |

**Learning and selection.** PerProp uses a nominal response and four perturbed responses per training example, with phase starts separated by 1.5 ms. A reference channel supplies 32 spikes through a weight of 16; paired perturbations change this weight by $\pm 8$. Feedback is estimated from exponentially averaged spike-response/current-response products and spike-response powers at rate $0.02$. This streaming implementation carries its accumulated statistics across examples. Its output loss is $\frac{1}{2}\sum_i[(c_i - t_i)/32]^2$, where $c_i$ is the measured spike count and the target $t_i$ is 24 for the correct class and 8 otherwise. Per-example PerProp weight increments are clipped to $[-1, 1]$ before accumulation and programming.

R-STDP uses baseline-subtracted causal correlation measurements, a spike-count reward, and an exponentially averaged reward baseline for each class. Its learning rate $0.03$ is selected by seed-0 validation from $\{0.003, 0.01, 0.03\}$, then held fixed for all three seeds. Each method trains for five epochs; the first 128 training examples collect learning statistics before changing forward weights. After each epoch we evaluate the validation set, retain the checkpoint with highest validation accuracy, and evaluate that checkpoint on the held-out test set. Seeds 0–2 share the same initial weights and training-example order across the two methods. The PPU measurement schedule and weight-update implementation are detailed in Appendix I.

### E.6 CROSSBAR SIMULATIONS

**Network and training.** The crossbar simulation uses a width-128 residual MLP with eight blocks, 656,640 trainable weights, and 18 forward arrays: an input projection, two arrays per residual block, and an output projection. Each residual branch has one neuron-response layer between its two arrays; the second array is initialized with a factor $1/\sqrt{16}$. The input projection has an additional neuron-response layer. We use the normalized, flattened CIFAR-10 data and fixed 45,000/5,000/10,000 split from Appendix E.2, without image augmentation.

All arms run 12,000 updates with batch size 128, SGD with momentum $0.9$, constant learning rate $0.01$, and zero weight decay. Training batches sample examples with replacement. Validation is

evaluated every 1,000 updates, and the final model is evaluated on the test set. Seeds 0–2 pair the network initialization, fixed hardware variations, and training batches. PerProp and Hybrid use perturbation size $\varepsilon = 0.1$ and one measurement per positive or negative neuron perturbation. PerProp uses 96 Gaussian feedback probes, quantized by the input converter, with feedback rate $0.1$, 20 initialization updates, and one feedback update after every forward-weight update.

**Component model.** Neuron $j$ has response

$$\phi_j(z) = g_j \tanh(a_j z + c_j) + d_j,$$
$$g_j, a_j \sim \mathcal{N}(1, 0.1^2), \qquad c_j \sim \mathcal{N}(0, 0.05^2), \qquad d_j \sim \mathcal{N}(0, 0.02^2).$$

These parameters are sampled independently and remain fixed during training. Table 14 specifies programming and measurement imperfections. Signed weight ranges describe effective conductances. For a value clipped to $[-R, R]$, the quantization interval is $2R/(2^b - 1)$ at $b$ bits. Forward weights are reprogrammed after each optimizer update; each PerProp feedback update is also quantized and written with independent noise.

Table 14: Component settings for the reported CIFAR-10 crossbar simulation. Noise draws are Gaussian; fixed neuron and measurement-channel variation is sampled once per seed.

| Component | Setting |
|---|---|
| Forward / feedback programming | 7 bits, signed range $[-2, 2]$ |
| Input and feedback-probe conversion | 8 bits, range $[-4, 4]$ |
| Forward / feedback write-noise std. | $0.01$ times mean absolute target weight |
| Gaussian feedback-probe amplitude | Unit standard deviation before conversion |
| Neuron-measurement gain variation | $\mathcal{N}(0, 0.02^2)$ around unit gain |
| Neuron-measurement offset | $\mathcal{N}(0, 0.01^2)$ |
| Neuron / forward-probe read-noise std. | $0.01$ |
| PerProp feedback / Hybrid reverse-read noise | $0.01$ times output RMS |
| Feedback-probe output conversion | Analog; no additional ADC quantization |
| Temperature | 300 K, fixed |
| Sample-and-hold droop / added hold noise | 0 / 0 |

The neuron-measurement channel returns $(1 + \gamma_j)\phi_j(z) + o_j + \nu$, where $\gamma_j$ and $o_j$ are the fixed gain and offset variations in Table 14, and $\nu$ is fresh measurement noise. PerProp learns and stores the quantized feedback state, including accumulated write error. Hybrid uses a reverse read of the forward arrays with the stated read noise. The BP reference accesses the simulator's true neuron derivatives and exact transposes of the programmed weights. The forward-weight optimizer runs digitally in this behavioral simulation. The proposed circuit mapping is described in Appendix J; component energy budgets and extrapolations are given in Appendix K.

# F LEARNING SCALABILITY

The scaling experiments test whether larger models and greater training resources continue to improve PerProp relative to BP and Hybrid. This appendix provides every completed configuration's score, the fitted slopes, and the operation-count calculation. The data, architecture, schedules, and evaluation window are specified in Appendix E.1.

## F.1 COMPLETE RESULTS AND SCALING PROTOCOLS

Table 15 contains all 20 configurations. Seventeen contribute to the main figure; three additional width settings are retained here. A shared six-layer, width-384 reference belongs to the data, batch, depth, and model-size sweeps. Counting that reference once avoids treating repeated use in a plot as an additional experiment. Each configuration has one run per arm.

The data sweep increases training updates at fixed architecture and batch size. The batch sweep fixes 6,480 updates, so its larger batches also process more tokens. The depth sweep fixes embedding width and batch size while increasing updates in proportion to depth and reducing the peak learning rate for the deeper networks. The model-size sweep changes architecture and training budget together. These are complementary training protocols; the depth result, for example, measures deeper networks trained for longer rather than an isolated change in layer count at fixed compute.

Table 15: All 20 scaling configurations. Each entry gives validation NLL in nats / perplexity, using the final evaluation window and one seed. The last three width settings are additional to the 17 configurations in the main figure.

| Configuration | BP | Hybrid | PerProp |
|---|---|---|---|
| Model size: $L = 3, d = 384$ | 4.4571 / 86.24 | 4.4469 / 85.36 | 4.9418 / 140.03 |
| Reference: $L = 6, d = 384$ | 4.0813 / 59.22 | 4.0902 / 59.75 | 4.4650 / 86.92 |
| Model size: $L = 12, d = 384$ | 3.7898 / 44.25 | 3.7879 / 44.16 | 4.1205 / 61.59 |
| Model size: $L = 12, d = 768$ | 3.2874 / 26.77 | 3.2745 / 26.43 | 3.6693 / 39.22 |
| Model size: $L = 24, d = 1024$ | 2.8746 / 17.72 | 2.9134 / 18.42 | 3.3014 / 27.15 |
| Data: $2.5N_{\text{budget}}$ | 5.5160 / 248.63 | 5.5208 / 249.84 | 5.9515 / 384.32 |
| Data: $5N_{\text{budget}}$ | 4.8996 / 134.24 | 4.8857 / 132.38 | 5.4836 / 240.72 |
| Data: $10N_{\text{budget}}$ | 4.3817 / 79.98 | 4.3759 / 79.51 | 4.9212 / 137.17 |
| Data: $40N_{\text{budget}}$ | 3.9084 / 49.82 | 3.9169 / 50.24 | 4.1790 / 65.30 |
| Data: $80N_{\text{budget}}$ | 3.7759 / 43.64 | 3.7861 / 44.08 | 4.0300 / 56.26 |
| Depth: 3 | 4.4599 / 86.48 | 4.4664 / 87.04 | 4.9661 / 143.47 |
| Depth: 12 | 3.8038 / 44.87 | 3.8150 / 45.38 | 4.1203 / 61.58 |
| Depth: 24 | 3.5952 / 36.42 | 3.6012 / 36.64 | 3.8126 / 45.27 |
| Batch: 8,192 tokens | 4.8590 / 128.90 | 4.8418 / 126.69 | 5.3431 / 209.16 |
| Batch: 16,384 tokens | 4.3671 / 78.81 | 4.3680 / 78.89 | 4.8670 / 129.93 |
| Batch: 65,536 tokens | 3.8903 / 48.92 | 3.8884 / 48.83 | 4.1177 / 61.42 |
| Batch: 131,072 tokens | 3.7416 / 42.16 | 3.7626 / 43.06 | 3.9596 / 52.44 |
| Width: 192 | 5.1563 / 173.51 | 5.1463 / 171.79 | 5.4681 / 237.02 |
| Width: 768 | 3.5593 / 35.14 | 3.5636 / 35.29 | 3.9802 / 53.53 |
| Width: 1,536 | 3.1635 / 23.65 | 3.1670 / 23.74 | 3.6690 / 39.21 |

## F.2 FITTED RATES AND PREDICTION QUALITY

For resource $x$ and validation perplexity $p$, the straight fits use

$$\log p = a + b \log x. \tag{24}$$

Since $\log p$ is validation NLL, this is also least-squares regression of NLL on $\log x$. Multiplying an axis by a constant changes its intercept but not its slope. Table 16 reports $|b|$ and the ratio of the PerProp and BP slope magnitudes. These are descriptive fits to the observed settings. One seed per setting does not provide seed-level uncertainty on these rates.

PerProp has steeper fitted improvement along the data, batch-size, and depth protocols, while the parameter and FLOP rates are close to BP's. The only exception additional width sweep behaves differently: at its fixed feedback-probe settings, PerProp improves a bit more slowly than BP and the NLL gap grows. Increasing scale is therefore not interchangeable across these protocols. The measured curves support sustained improvement and narrowing gaps on the specified axes; their slopes do not establish an asymptotic crossing or equality of limiting performance. The largest model still has perplexity 27.15 for PerProp, 17.72 for BP, and 18.42 for Hybrid.

Table 16: Absolute least-squares slopes of $\log(\text{perplexity})$ against $\log(\text{resource})$. The last column divides the PerProp slope magnitude by the BP slope magnitude.

| Axis | BP | Hybrid | PerProp | Ratio to BP |
|---|---|---|---|---|
| Training data | 0.4936 | 0.4891 | 0.5761 | 1.167 |
| Batch size | 0.3912 | 0.3806 | 0.5073 | 1.297 |
| Depth | 0.4143 | 0.4142 | 0.5490 | 1.325 |
| Parameters | 0.3842 | 0.3762 | 0.3902 | 1.016 |
| Estimated FLOPs | 0.2264 | 0.2216 | 0.2289 | 1.011 |
| Additional width | 0.9378 | 0.9326 | 0.8486 | 0.905 |

Where a displayed fit extends past the observations, that segment is an extrapolation of the fitted function. Markers correspond only to measured runs. Neither the fitted extensions nor repeated appearances of a run in different panels add measurements.

### F.3 Estimated training FLOPs

The FLOP panel uses the same five model-size runs, with arm-specific costs. We count a multiply–accumulate as two floating-point operations and use a common full-attention convention. This makes the estimate independent of causal-tile skipping or implementation-specific kernel fusion. Forward, input-error, and weight-update matrix products have the same dimensions in BP, Hybrid, and PerProp.

Let $P_f$ be the number of weights in the learned linear maps, including the vocabulary head; let $Q$ be the sum of their input dimensions. With $m$ probes, one dense feedback refresh has two matrix products: computing the forward responses and accumulating probe–response correlations. Their combined cost is $4mP_f$ FLOPs. The implemented denominator, normalization, and EMA arithmetic add $(3m-1)Q + 3P_f$. If $K$ optimizer updates run on $R$ data-parallel replicas, including $K_0 = 20$ initialization refreshes, the feedback estimate is

$$F_{\text{feedback}} = R(K + K_0)\left[4mP_f + (3m-1)Q + 3P_f\right]. \tag{25}$$

The per-replica factor counts the feedback computations actually executed on each rank. The neuron term adds the two perturbed evaluations and their difference arithmetic using the collector's unit-cost scalar convention. BP retains its ordinary nonlinear operation budget; Hybrid and PerProp receive the measured-slope term. Table 17 gives the resulting totals.

Table 17: Estimated total training FLOPs ($10^{18}$ operations) for the five model-size runs. The overhead column compares PerProp with BP at the executed budgets.

| Parameters (M) | BP | Hybrid | PerProp | Overhead (%) |
|---|---|---|---|---|
| 44.302 | 0.01727 | 0.01728 | 0.01732 | 0.29 |
| 49.612 | 0.04438 | 0.04442 | 0.04450 | 0.25 |
| 60.234 | 0.12817 | 0.12832 | 0.12845 | 0.22 |
| 162.935 | 1.45744 | 1.45861 | 1.46192 | 0.31 |
| 407.588 | 14.74183 | 14.75292 | 14.79155 | 0.34 |

These are algorithmic FLOP estimates, not hardware instruction counts or measured elapsed time. They omit optimizer operations, evaluation, random-number generation, communication, and memory traffic. The same implementation can have a different runtime ratio because these operations and kernel utilization affect elapsed time. The separate RTX A6000 comparison in Appendix G measures training-step time directly. Projected training energy additionally accounts for learning-circuit and storage operations, as specified in Appendix K.4.

# G COMPARISONS WITH OTHER BACKPROPAGATION ALTERNATIVES

These comparisons test predictive performance and training efficiency against existing alternatives to backpropagation. A learning rule specifies how the network weights are updated. We use language models to compare learning and transfer across corpora, and retain the CIFAR-10 comparisons below. Architecture, data, optimizer settings, and learning-rate selection are specified in Appendix E.2.

## G.1 IMPLEMENTED LEARNING RULES

**Feedback-based rules.** Feedback alignment uses fixed random feedback at the selected affine maps (Lillicrap et al., 2016). Direct feedback alignment instead supplies hidden teaching signals through fixed projections of the output error (Nøkland, 2016). The reported GPT DFA uses full layerwise projections at hidden affine responses, embeddings, and normalization sites; its 194 feedback matrices are shared across token positions. An earlier blockwise GPT DFA retained within-block backpropagation and is not the reported DFA result. The Mamba-2 implementation injects direct signals at its configured neuron interfaces while retaining local arithmetic within the surrounding structural operations. DRTP uses the negative one-hot target in place of the output-error vector (Frenkel et al., 2021).

PEPITA computes a nominal pass and a second pass whose input is modulated by a fixed projection of the output error (Dellaferrera & Kreiman, 2022). Hidden updates use the difference in local activities, while the output uses the supervised error. For GPT, the perturbation is applied to token embeddings rather than discrete token indices. Difference target propagation learns inverse maps for the MLP blocks and uses difference-corrected targets (Lee et al., 2015). Forward–forward uses its label-conditioned ReLU MLP with local goodness objectives and normalized interlayer inputs (Hinton, 2022); its architecture exception is specified in Table 8.

**Perturbation-based rules.** Weight perturbation / MeZO evaluates symmetric parameter displacements and forms a random-direction gradient estimate (Malladi et al., 2023). It uses 64 directions per update, except GPT, which uses 16. The node-perturbation comparison perturbs block interfaces conceptually: one scalar directional loss signal combines contributions from all interfaces, and each interface receives its own direction multiplied by that signal. The INP-like comparison instead projects each interface's input error separately, removing the cross-interface terms. Both use 64 Rademacher directions.

For NP and the INP-like comparison, the code evaluates these estimators in an oracle directional-derivative form. A first exact backward calculation obtains interface errors; a second pass injects their sampled projections to form local weight updates. Consequently their reported scores and timings describe this two-pass software implementation, rather than a physical finite-difference execution of all required probes. The term INP identifies the per-interface perturbation comparison in the CIFAR table; its exact implementation is the projection described here.

OPZO learns direct feedback from correlations between injected activity noise and output changes (Xiao et al., 2024). Our architecture adapter perturbs the configured block inputs in one additional noisy forward pass, correlates the noise with the change in logits, and averages the resulting direct-feedback map. PseuZO uses its sliding history of input/response changes to estimate the relevant local operator (Yue et al., 2025); PseuZO-NP supplies its history using node perturbations. They use window length 16 and momentum $0.9$.

**Coordinate-based comparisons.** BOND adapts coordinate perturbation magnitudes using its moment statistics (Clark et al., 2026). The retained ResMLP runs use the reservoir partition, which sweeps the complete input to each queried interface. HZO recursively forms intermediate targets through coordinate queries, then uses local autodifferentiation at the leaves (Cao et al., 2026). Query batching groups evaluations without reducing their number. Their ResMLP results are reported below; the larger interface sizes make the other screened architectures substantially more expensive.

## G.2 LANGUAGE-MODEL RESULTS

Table 18 reports the final WikiText-103 validation NLL and frozen-checkpoint TinyStories evaluation. The same model trained on WikiText supplies both columns for each architecture; TinyStories adds no optimization or model selection. The loss is measured per token under the shared WikiText tokenizer and evaluation protocol in Appendix E.2.

Among the compared alternatives, PerProp has the lowest NLL on both corpora and both architectures. BP and Hybrid provide references with access to transposed forward weights. GPT OPZO has two completed runs; its seed-0 run diverged at update 1,259.

Table 18: Language-model validation NLL (nats, lower is better). WikiText-103 is the training corpus; TinyStories is evaluated without further training. Entries use three seeds except GPT OPZO, which uses its two completed seeds; the third diverged.

| Rule | WikiText GPT | WikiText Mamba-2 | TinyStories GPT | TinyStories Mamba-2 |
|---|---|---|---|---|
| BP | 4.915 ± 0.022 | 4.461 ± 0.017 | 7.011 ± 0.036 | 6.609 ± 0.075 |
| Hybrid | 4.915 ± 0.022 | 4.222 ± 0.070 | 7.011 ± 0.036 | 6.512 ± 0.117 |
| PerProp | 5.579 ± 0.023 | 4.546 ± 0.019 | 7.177 ± 0.019 | 6.653 ± 0.059 |
| Feedback alignment | 6.314 ± 0.043 | 9.084 ± 0.057 | 7.564 ± 0.029 | 9.118 ± 0.032 |
| Direct feedback alignment | 6.121 ± 0.009 | 5.677 ± 0.010 | 7.433 ± 0.023 | 7.337 ± 0.006 |
| DRTP | 6.679 ± 0.009 | 6.303 ± 0.011 | 7.579 ± 0.006 | 7.467 ± 0.012 |
| PEPITA | 6.268 ± 0.008 | div. | 7.472 ± 0.012 | div. |
| Weight perturbation / MeZO | 7.590 ± 0.011 | 7.357 ± 0.009 | 8.102 ± 0.027 | 7.997 ± 0.025 |
| Node perturbation | 6.965 ± 0.013 | div. | 7.676 ± 0.056 | div. |
| OPZO | 6.894 ± 0.116 | 6.262 ± 0.010 | 7.775 ± 0.191 | 7.483 ± 0.046 |

The language-model table omits methods whose earlier screening failed to produce a completed result. BOND and HZO exceeded the time budget even at eight layers; the estimated full ConvNeXt runs required about 25 and 29 days. PseuZO and PseuZO-NP diverged in language-model screening. Difference target propagation, forward–forward, and the INP-like implementation have no defined adapter for these language-model architectures. In tables, `n/a` denotes an undefined rule for the specified architecture, `div.` a failed numerical training run, and `t.o.` an unfinished time-limited run. A failure at the smaller screening configuration is not a measured runtime or score for the 24-layer replacement model.

### G.3 CIFAR-10 RESULTS

We compare learning rules on ResMLP-16 and ConvNeXt V2 for CIFAR-10 classification (Krizhevsky, 2009) (Table 19).

Table 19: CIFAR-10 test accuracy (%, ↑), reported as mean ± sample standard deviation over three seeds. "div.": divergence; "n/a": undefined rule; "t.o.": time limit.

| Learning rule | ResMLP-16 | ConvNeXt V2 |
|---|---|---|
| BP | 56.40 ± 0.24 | 83.13 ± 0.19 |
| Hybrid | 56.16 ± 0.34 | 83.12 ± 0.22 |
| **PerProp** | **54.75** ± 0.58 | **80.86** ± 0.51 |
| Feedback alignment | 47.39 ± 0.39 | 70.17 ± 0.44 |
| Direct feedback alignment | 50.97 ± 0.99 | 62.69 ± 0.92 |
| DRTP | div. | 10.00 ± 0.00 |
| PEPITA | 48.34 ± 0.19 | 15.25 ± 4.85 |
| Difference target propagation | 50.80 ± 0.59 | n/a |
| Forward-forward | 39.94 ± 0.14 | n/a |
| Weight perturbation / MeZO | 43.29 ± 0.43 | 33.40 ± 0.41 |
| Node perturbation | 49.15 ± 0.60 | 47.76 ± 1.11 |
| INP | 51.69 ± 0.41 | n/a |
| OPZO | 51.83 ± 0.49 | 52.72 ± 0.59 |
| BOND | 56.31 ± 0.33 | t.o. |
| HZO | 56.62 ± 0.32 | t.o. |
| PseuZO | 37.16 ± 0.61 | 36.77 ± 0.63 |
| PseuZO-NP | 41.04 ± 0.97 | 20.47 ± 0.69 |

BOND and HZO achieve accuracy comparable to BP on ResMLP. Their coordinate sweeps require more evaluations as the queried interface grows, accounting for the long projected ConvNeXt runs described above (Clark et al., 2026; Cao et al., 2026). Figure 6 shows the resulting accuracy–time trade-off.

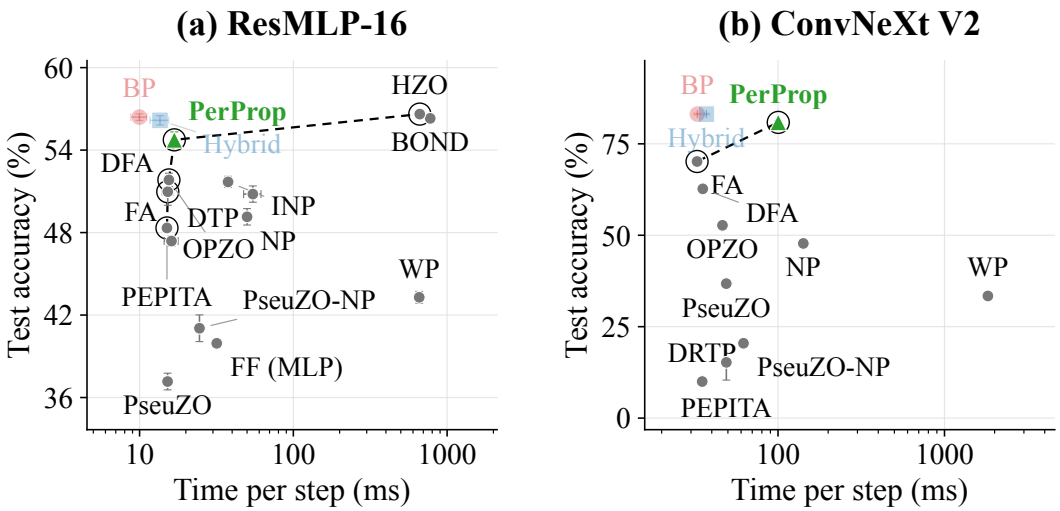


Figure 6: Pareto frontiers of CIFAR-10 accuracy and training time per step on an RTX A6000. BP and Hybrid are faded references excluded from the frontier because they use transposed forward weights.

### G.4 Training time and Pareto construction

Table 20 reports steady training-step time on one RTX A6000, measured in separate sequential benchmarks using the protocol in Appendix E.2. The interval includes the learning rule, optimizer, and feedback refresh. Data loading, evaluation, initialization, and checkpoint writing are outside it.

Table 20: Median training-step time (ms) on one RTX A6000. Scores in the paper come from full training runs; these times come from separate short, sequential benchmarks. Missing entries retain the experiment status: divergence (div.), time limit (t.o.), or undefined rule (n/a).

| Rule | ResMLP-16 | ConvNeXt V2 | GPT | Mamba-2 |
|---|---|---|---|---|
| BP | 9.97 | 32.76 | 169.73 | 180.81 |
| Hybrid | 13.58 | 37.12 | 206.03 | 437.98 |
| PerProp | 16.82 | 100.35 | 209.74 | 449.56 |
| Feedback alignment | 16.13 | 32.63 | 127.82 | 169.84 |
| Direct feedback alignment | 15.25 | 35.32 | 514.75 | 434.64 |
| DRTP | div. | 35.02 | 184.39 | 417.92 |
| PEPITA | 15.07 | 48.93 | 230.66 | div. |
| Weight perturbation / MeZO | 658.46 | 1822.55 | 1870.45 | 8393.27 |
| Node perturbation | 50.02 | 141.86 | 598.11 | div. |
| OPZO | 15.51 | 46.37 | 278.44 | 746.06 |
| INP-like projection | 37.62 | n/a | n/a | n/a |
| Difference target propagation | 54.53 | n/a | n/a | n/a |
| Forward–forward | 31.75 | n/a | n/a | n/a |
| BOND | 776.13 | t.o. | t.o. | t.o. |
| HZO | 663.99 | t.o. | t.o. | t.o. |
| PseuZO | 15.21 | 48.86 | div. | div. |
| PseuZO-NP | 24.52 | 62.02 | div. | div. |

A point is Pareto-dominated if another eligible rule has no larger training-step time and no worse predictive performance, with at least one strict improvement. The language-model frontiers minimize both time and NLL; the vision frontiers minimize time and maximize accuracy. BP and Hybrid appear as faint references and are excluded from the alternative-method frontier because they use forward-weight transposes. The earlier blockwise GPT DFA is excluded from the displayed comparison. The same step time is paired with both WikiText and TinyStories scores because they evaluate the same trained checkpoint.

NP and the INP-like comparison include their oracle-replay execution in these times; coordinate-query methods include their actual query batching.

# H NON-DIFFERENTIABLE NETWORK EXPERIMENTS

These experiments test whether the same measurement-based approach can train networks with several non-differentiable neuron types. The comparison is a test of trainability across responses: each task has its own suitable surrogate baseline. We report the full three-arm results and tuning controls here; Appendix E gives architectures and training settings.

## H.1 SPIKING-NETWORK RESULTS

MNIST uses rate-coded input spikes and two hidden layers of 256 neurons, evaluated with LIF, QIF, EIF, and Izhikevich-form dynamics. SHD supplies recorded auditory spike events to one recurrent layer of 256 ALIF neurons. The readout is a non-spiking leaky integrator in both cases. The state equations and time discretization are listed in Appendix E.3.

Surrogate BPTT uses a chosen derivative for the threshold while retaining the neuron-state and reset computations in its temporal graph. Hybrid measures the local neuron-state transition and uses transposed synaptic weights. PerProp uses those measurements with spike-response feedback learning as detailed in Appendix C.3. Table 21 reports final-epoch accuracy, and Table 24 retains individual seed results. The measured methods do not uniformly outperform every surrogate; the common update can train all of the tested responses without choosing an analytical derivative for each one.

Table 21: Test accuracy (%) for networks with non-differentiable units. Values are means and sample standard deviations over three seeds. BP uses the corresponding surrogate; the EIF surrogate and PerProp settings are validation-selected.

| Units | BP surrogate | Hybrid | PerProp |
|---|---|---|---|
| Comparator | 47.40 ± 0.32 | 48.58 ± 0.28 | 48.59 ± 0.57 |
| Probabilistic bit | 47.17 ± 0.31 | 48.49 ± 0.60 | 48.35 ± 0.16 |
| LIF | 97.25 ± 0.17 | 97.46 ± 0.21 | 97.04 ± 0.33 |
| QIF | 97.23 ± 0.16 | 97.48 ± 0.31 | 97.09 ± 0.11 |
| EIF | 96.54 ± 0.32 | 97.20 ± 0.31 | 97.04 ± 0.09 |
| Izhikevich | 97.22 ± 0.05 | 97.24 ± 0.20 | 96.89 ± 0.15 |
| Recurrent ALIF | 86.90 ± 0.93 | 86.84 ± 0.38 | 86.44 ± 0.50 |

## H.2 EIF TUNING AND THE FIXED-SURROGATE REFERENCE

We tune both the conventional surrogate and PerProp on a stratified 6,000-example holdout from MNIST training data. Final test scores use fresh full-training-set runs with the selected settings. The surrogate search screens 70 configurations over kernel family, width, learning rate, and selected amplitude gains. The PerProp search screens 54 configurations over perturbation magnitude, learning rate, and feedback rate. Candidate replication uses seeds 0–2; mean final validation accuracy selects the configuration, with validation cross-entropy breaking ties. Test labels play no part in selection.

Table 22: EIF test accuracy after the original and validation-selected settings. Every row trains on all 60,000 MNIST training examples for three epochs and evaluates on the 10,000-example test set.

| Rule | Setting | Test accuracy (%) |
|---|---|---|
| Original surrogate | Arctangent, $a = 2$, LR 0.001 | $93.90 \pm 0.43$ |
| Selected surrogate | Rectangular, $a = 2$, LR 0.001 | $96.54 \pm 0.32$ |
| Original PerProp | $\varepsilon = 1$, $\rho = 0.2$, LR 0.001 | $96.96 \pm 0.16$ |
| Selected PerProp | $\varepsilon = 0.5$, $\rho = 0.5$, LR 0.001 | $97.04 \pm 0.09$ |

The previously reported 93.90% reference is the original fixed arctangent setting, not the best setting selected by the sweep. Tuning raises its mean to 96.54%. The main comparison uses this selected baseline and the selected PerProp setting. Appendix E.3 defines all kernels and search ranges, including their common amplitude convention. A rectangular surrogate can match a finite-difference threshold slope away from endpoints, but the complete rules still differ: PerProp measures the reset and state transition and learns feedback.

## H.3 COMPARATOR AND PROBABILISTIC-BIT NETWORKS

In a comparator network, each neuron compares its weighted input with a reference threshold and returns $+1$ or $-1$. The comparison is between the summed input and the threshold, rather than between two data examples. A probabilistic-bit network has the same binary outputs, but draws $+1$ with input-dependent probability $\sigma((z - \theta_j)/T)$. In both settings, the realized response is discrete. We use ResMLP-16 with a fixed threshold $\theta_j$ drawn independently for each neuron; the learning method is not given these thresholds.

The standard comparator baseline uses a clipped straight-through derivative centered on the nominal threshold. The probabilistic-bit baseline differentiates the nominal mean response. PerProp and Hybrid instead measure the actual responses at positive and negative input offsets. For probabilistic bits, nominal and perturbed evaluations share the same uniform random sample, preserving the correct marginals while reducing difference noise. Its random sample can still change the output when the perturbation crosses the corresponding probability level.

Table 23 supplies additional baselines whose surrogate is centered on the actual threshold. These controls separate knowing the fixed threshold from estimating the response through measurements. They retain the same data, architecture, training budget, and three seeds as the nominal-surrogate comparison.

Table 23: Threshold-aware surrogate controls: test accuracy (%) with the actual fixed neuron thresholds supplied to the surrogate.

| Network | Accuracy |
|---|---|
| Comparator | 48.31 $\pm$ 0.09 |
| Probabilistic bit | 47.83 $\pm$ 0.28 |

Table 24: Individual test accuracies (%) underlying Table 21.

| Units | Rule | Seed 0 | Seed 1 | Seed 2 |
|---|---|---|---|---|
| Comparator | BP surrogate | 47.04 | 47.48 | 47.67 |
| Comparator | Hybrid | 48.60 | 48.29 | 48.84 |
| Comparator | PerProp | 48.96 | 47.93 | 48.88 |
| Probabilistic bit | BP surrogate | 47.27 | 47.41 | 46.82 |
| Probabilistic bit | Hybrid | 49.11 | 47.91 | 48.44 |
| Probabilistic bit | PerProp | 48.18 | 48.38 | 48.49 |
| LIF | BP surrogate | 97.18 | 97.13 | 97.44 |
| LIF | Hybrid | 97.40 | 97.28 | 97.69 |
| LIF | PerProp | 97.13 | 96.67 | 97.31 |
| QIF | BP surrogate | 97.40 | 97.09 | 97.21 |
| QIF | Hybrid | 97.53 | 97.14 | 97.76 |
| QIF | PerProp | 97.18 | 96.97 | 97.11 |
| EIF | BP surrogate | 96.76 | 96.69 | 96.18 |
| EIF | Hybrid | 97.14 | 96.92 | 97.53 |
| EIF | PerProp | 97.14 | 96.98 | 96.99 |
| Izhikevich | BP surrogate | 97.18 | 97.20 | 97.28 |
| Izhikevich | Hybrid | 97.02 | 97.41 | 97.28 |
| Izhikevich | PerProp | 97.06 | 96.83 | 96.79 |
| Recurrent ALIF | BP surrogate | 87.81 | 86.93 | 85.95 |
| Recurrent ALIF | Hybrid | 87.23 | 86.79 | 86.48 |
| Recurrent ALIF | PerProp | 86.22 | 86.09 | 87.01 |

The PerProp comparator and probabilistic-bit results use the current 96-probe Gaussian feedback update, with rate $0.1$ and 20 initialization refreshes. The SNN rows use the measured-spike variant described in Appendix C.3; none of the reported PerProp rows uses an exact measured transpose.

# I BRAINSCALES-2 IMPLEMENTATION AND EXPERIMENTS

The BrainScaleS-2 experiment tests whether PerProp can update both layers of a physical spiking network using the measurements and processors available on the chip. This appendix describes the measured signals, their conversion into weight updates, and the matched R-STDP comparison. Data generation, encoding, and numerical settings are given in Appendix E.5.

## I.1 ANALOG NEURONS, MEASUREMENTS, AND THE PPU

BrainScaleS-2 (Figure 7) implements membrane and synaptic dynamics with analog circuits. Incoming events deliver synaptic currents, membrane voltages evolve continuously, and threshold crossings generate output spikes. Programmable synaptic weights determine the effect of each event. The chip combines these analog neurons with digital event routing and a programmable Plasticity Processing Unit (PPU) that can read measurements and update synaptic weights (Pehle et al., 2022).

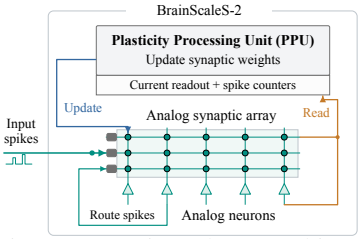


Figure 7: BrainScaleS-2 architecture (Pehle et al., 2022).

Our network uses five input channels, 32 hidden analog neurons, and three output analog neurons. Learning uses spike counters and column analog-to-digital converter (CADC) measurements configured to read the excitatory synaptic-input response. The PPU holds feedback statistics, fractional weight states, and the learning kernel. It computes output errors, learns feedback, propagates errors to the hidden layer, and programs both synaptic layers. The host supplies encoded examples and labels, schedules experiments, saves states, and evaluates predictions. It does not compute the learning updates. Neural dynamics are analog; the learning arithmetic runs on the chip's digital PPU.

## I.2 PAIRED MEASUREMENT WINDOWS

For each training example, PerProp performs the five windows in Table 25. The stimulus and label are held fixed across them. Neuron state and spike counters are reset at each phase. A reference input supplies 32 spikes through weight 16; adding or subtracting 8 from that reference weight produces the perturbation. Independent random signs choose the hidden-neuron perturbation directions. Each phase starts 1.5 ms after the preceding one within the 20 ms training period.

Table 25: Measurements for one PerProp training example on BrainScaleS-2. $c$ denotes spike counts and $A$ the averaged analog synaptic-response readout.

| Phase | Perturbation | Recorded quantities and purpose |
|---|---|---|
| 0 | Nominal reference | Hidden and output counts for prediction, loss, and presynaptic activity. |
| 1 | Positive output reference | Output count and current response for postsynaptic sensitivity. |
| 2 | Negative output reference | Paired output sensitivity measurement. |
| 3 | Signed hidden reference | Hidden count/current and downstream current responses for sensitivity and feedback. |
| 4 | Opposite hidden reference | Paired hidden sensitivity and feedback measurements. |

The CADC readout is averaged between 0.425 and 1.050 ms after each phase starts. An accepted phase has at least eight ADC samples and unsaturated spike counters. The 0.5 ms input spike window and the measurement schedule are specified by the executed kernel; they are physical chip times, rather than simulated biological times.

## I.3 MEASURED SENSITIVITIES AND LEARNED FEEDBACK

Let $c^h, c^o$ denote nominal hidden and output spike counts. For an output neuron $i$, define its paired count and analog-response differences by

$$dc_i^o = c_i^{o,+} - c_i^{o,-}, \qquad dA_i^o = A_i^{o,+} - A_i^{o,-}.$$

The kernel measures both the count sensitivity and the reference-input gain,

$$\chi_i^o = \frac{dc_i^o}{dA_i^o}, \qquad \gamma_i^o = \frac{dA_i^o}{2\Delta w\, n_{\mathrm{ref}}}, \qquad \Delta w = 8, \quad n_{\mathrm{ref}} = 32. \tag{26}$$

It sets $\chi_i^o = 0$ when $|dA_i^o| \le 0.1$ or $dc_i^o \le 0$. This avoids dividing by a negligible response and uses the increasing branch of the measured count response.

For hidden neuron $j$, let $u_j \in \{-1, +1\}$ be its perturbation sign, $q_j = c_j^{h,+} - c_j^{h,-}$ its spike-count difference, and $dA_j^h$ its current-response difference. The corresponding factors are

$$\chi_j^h = \frac{q_j}{dA_j^h}, \qquad \gamma_j^h = \frac{dA_j^h}{2\Delta w\, n_{\mathrm{ref}}\, u_j}. \tag{27}$$

The kernel sets $\chi_j^h = 0$ when $|dA_j^h| \le 0.1$ or $q_j u_j \le 0$. The gain factors convert the measured analog-response coordinates to the count-times-weight coordinates used to program the synapses.

Let $y_i = A_i^{o,h+} - A_i^{o,h-}$ be the downstream analog response to the hidden perturbation pair. The PPU maintains one power $P_j$ per hidden neuron and one correlation $R_{ji}$ per feedback connection:

$$\begin{aligned} P_j &\leftarrow (1-\rho)P_j + \rho q_j^2, \\ R_{ji} &\leftarrow (1-\rho)R_{ji} + \rho q_j y_i, \qquad \rho = 0.02. \\ B_{ji} &\leftarrow R_{ji}/P_j \quad \text{if } P_j > 0.25, \end{aligned} \tag{28}$$

Otherwise the previous $B_{ji}$ is retained. Statistics start at zero, and the first 128 training examples collect feedback statistics without changing forward weights. These examples are part of the first epoch. Feedback subsequently updates after each example.

Equation (28) is the streaming normalized-correlation implementation: it averages numerator and denominator before taking their ratio. The dense rule in (7) averages already normalized probe-batch estimates. Both learn feedback from input–response correlations, but the two finite-sample statistics are different. On the chip, $B$ maps an output-current error to a hidden-count error and incorporates the measured downstream synaptic gain. It is not obtained by reading or copying the programmed forward weights.

### I.4 Weight updates on the chip

The count target is $t_i = 24$ for the correct class and 8 for the others. The kernel uses

$$e_i = \frac{c_i^o - t_i}{32}, \qquad \delta_i^o = \chi_i^o e_i, \qquad g_j^h = \sum_i B_{ji}\delta_i^o. \tag{29}$$

It logs $\frac{1}{2}\sum_i[(c_i^o - t_i)/32]^2$ as the count loss. The error convention in (29) differs from that loss's derivative by a common factor 32, absorbed into the learning-rate convention.

For the hidden-to-output weight $W_{ij}^2$ and input-to-hidden weight $W_{jk}^1$, with input spike count $n_k$, the proposed increments are

$$\begin{aligned} dW_{ij}^2 &= -\eta\, c_j^h\, \gamma_i^o \chi_i^o e_i, \\ dW_{jk}^1 &= -\eta\, n_k\, \gamma_j^h \chi_j^h g_j^h. \end{aligned} \tag{30}$$

Each product uses presynaptic activity, measured postsynaptic sensitivity, and a received error. The gain calibration is included with the sensitivity. The PPU clips each increment to $[-1, 1]$, adds it to a fractional weight accumulator, clips that accumulator to $[0, 63]$, and rounds to the nearest integer for programming. Retaining fractional accumulators allows repeated sub-integer updates to change six-bit synapses. Both layers are trained throughout the five epochs after the initial warmup.

This implementation uses finite-window spike counts and measured current sensitivities. It does not reconstruct a time-unrolled digital neuron model. Its numerical settings therefore differ from the GPU SNN experiments, whose measured transitions propagate errors through stored time steps.

### I.5 Reward-modulated STDP baseline

R-STDP is a prior on-chip learning rule with a public BrainScaleS-2 task-training example (Wunderlich et al., 2019). The published demonstration trains a Pong task. We adapt its reward-modulated causal-correlation principle to the same Yin–Yang network, examples, initial weights, calibration, training order, epoch budget, and checkpoint selection used for PerProp.

For a synapse, let $A^{\mathrm{base}}$ and $A^{\mathrm{causal}}$ be its baseline and current correlation-sensor readings. The implementation uses eligibility

$$E_{ij} = \mathrm{clip}\big(\tfrac{1}{2}(A_{ij}^{\mathrm{base}} - A_{ij}^{\mathrm{causal}}), 0, 127\big)\,.$$

For label $y$, output counts define reward $R = [2c_y^o - \sum_{i\neq y} c_i^o]/32$. A separate running reward baseline $\bar{R}_y$ for each class is updated at rate $0.05$. Weight changes combine the reward advantage

and a rate-homeostasis term,

$$dW_{ij} = \eta(R - \bar{R}_y)E_{ij} + \frac{\lambda_h(r_i^* - c_i)}{5000 r_i^*}, \tag{31}$$

with $\lambda_h = 1$, target count 16 for hidden neurons, and $40/3$ for output neurons. Exploration perturbs the reference input around weight 16 with initial amplitude 8, decaying linearly over training. The PPU generates the perturbation from a sum of uniform random variables and clips reference weights to $[0, 63]$. R-STDP uses one activity window per example; PerProp uses the five windows above. Both use the same 20 ms example period. Their six-bit weights retain fractional accumulators.

Seed-0 validation selects learning rate $0.03$ from $\{0.003, 0.01, 0.03\}$, and this setting is fixed for all three seeds. Table 26 gives the validation-selected epoch and held-out test score for each run. Mean test accuracy is $62.1 \pm 1.8\%$ for PerProp and $36.9 \pm 1.8\%$ for R-STDP. Uniform random prediction has expected accuracy $1/3$ on this three-class task; this is an analytical chance reference, not another trained model.

Table 26: BrainScaleS-2 seed results on Yin–Yang. Checkpoints are selected by validation accuracy after each epoch; the selected checkpoint is then evaluated on the test set.

| Seed | Rule | Selected epoch | Validation (%) | Test (%) |
|---|---|---|---|---|
| 0 | PerProp | 1 | 64.0 | 61.9 |
| 0 | R-STDP | 2 | 37.3 | 39.0 |
| 1 | PerProp | 2 | 67.9 | 64.0 |
| 1 | R-STDP | 2 | 42.8 | 35.5 |
| 2 | PerProp | 4 | 62.3 | 60.4 |
| 2 | R-STDP | 3 | 36.9 | 36.3 |

### I.6 Investigation of an additional three-factor baseline

We also investigated ETLP (Quintana et al., 2024). We reproduced its learning updates from the reference implementation and verified their execution on the PPU using supplied neuron states. In our implementation, live spike-trace measurements lacked the required accuracy, and update computation was too slow to follow the analog neurons' dynamics. These constraints prevented a complete BrainScaleS-2 training result for ETLP.

The PPU replay used 96 supplied time steps with nine teaching events. It verified the update state against the reference computation, with maximum arithmetic times of 4.572 ms at a teaching event and 0.987 ms otherwise, before live measurement and weight programming. A separate live measurement test obtained a median analog-read time of 3.582 $\mu$s, but its held-out relative RMS trace error was 1.456 over 26 active channels. These diagnostics characterize the attempted implementation; they supply no ETLP classification score. The reproduction materials retain the reference comparison and measurement checks separately from the completed PerProp/R-STDP experiments.

# J CROSSBAR IMPLEMENTATION AND SIMULATION

The crossbar study tests learning when neuron responses and weight programming depart from their nominal mathematical models. It also provides the architecture used for the component energy projections. We distinguish the behavioral simulation, which supplies the accuracy results, from the proposed circuit implementation used for energy accounting.

## J.1 WHAT A CROSSBAR COMPUTES

A resistive crossbar contains a programmable conductance at each crossing of an input wire and an output wire. Applying input voltages $v_j$ produces currents proportional to conductance times voltage. The currents arriving on an output wire add, giving an analog matrix–vector product. Signed weights can be represented by the difference of two conductances or by a differential readout. Output sensing and neuron circuits convert the summed currents into the next layer's activity. Conversion, sensing, and interconnect energy are part of the effective MVM cost used in Appendix K.

Some arrays can be driven and sensed in the opposite direction to read the transposed map. This motivates Hybrid: it combines direct reverse-array reads with the same measured neuron slopes as PerProp. The PerProp architecture instead has a separate feedback array that stores learned $B$. Both variants use the nominal forward arrays for inference; their learning paths differ.

## J.2 MAPPING THE UPDATE TO CIRCUIT OPERATIONS

During the nominal forward computation, each array evaluates $Wa$ and each neuron produces its response. Sample-and-hold circuits retain activities and neuron inputs. Two additional neuron evaluations at $z \pm \varepsilon$ provide the slope through a differential readout. The arriving error is multiplied by this slope and sent through the feedback array, or through a reverse read of the forward array for Hybrid. Presynaptic activities and local errors form the outer-product weight increments.

To learn feedback, Gaussian probe inputs are driven through the forward array. Each input line accumulates its probe power $\sum_r \xi_{r,j}^2$, while correlation circuits accumulate $\sum_r \xi_{r,j} q_{r,i}$. A reciprocal shared along an input coordinate normalizes its correlation values. The normalized target is averaged with the retained feedback state and programmed into the separate array. These are the operations behind the feedback diagram: accumulation, normalization, temporal averaging, and programming. They implement the stochastic estimator of (7); no complete coordinate sweep is used.

The proposed circuit therefore needs signal storage, signed correlation accumulation, normalization, and forward/feedback weight programming in addition to the arrays and neurons. Appendix K assigns explicit budgets to these operations. The accuracy experiment evaluates the array and measurement models in software and uses a digital SGD optimizer for the forward weights.

## J.3 BEHAVIORAL SIMULATION AND CONTROLS

The simulated network is an eight-block residual MLP of width 128 with 656,640 weights and 18 forward arrays. Each neuron has a fixed gain, input scale, threshold offset, and output offset. Its response is $g_j \tanh(a_j z + c_j) + d_j$, with the distributions in Table 14. Forward and feedback weights are programmed at seven bits with write noise. Inputs and feedback probes use eight-bit conversion; neuron and synaptic reads include measurement noise. PerProp updates the learned feedback after every optimizer step and includes feedback quantization and programming error.

Hybrid uses a noisy reverse read of the programmed forward arrays. The BP control has access to the exact transpose of those programmed weights and to the derivative of the actual varied neuron response. It is an oracle available in the simulator. All three methods see the same forward-network imperfections and paired data/initialization seeds; the access used to compute their updates differs. The full setting is in Appendix E.6.

Table 27: CIFAR-10 accuracy (%) under the crossbar imperfections in Table 14. PerProp uses the replacement stochastic learned-feedback runs.

| Rule | Seed 0 | Seed 1 | Seed 2 | Mean ± SD |
|---|---|---|---|---|
| BP oracle | 44.19 | 44.05 | 44.11 | 44.12 ± 0.07 |
| Hybrid | 44.25 | 44.78 | 44.64 | 44.56 ± 0.27 |
| PerProp | 41.45 | 42.73 | 42.83 | 42.34 ± 0.77 |

The simulation logs feedback alignment against the current programmed forward weights in the same tensor orientation. Final means of layerwise cosine alignment are 0.709, 0.718, and 0.734 for seeds 0, 1, and 2; corresponding mean relative Frobenius errors are 0.661, 0.655, and 0.647. The accuracy results therefore use approximate learned feedback under these imperfections.

# K ENERGY ESTIMATION

We estimate energy by multiplying operation counts by component costs, following the component-based extrapolation approach of Wang & Achour (2026). The calculation includes forward computation, backward error propagation, neuron measurements, forward-weight updates, and feedback learning. Published MVM measurements and design projections supply alternative arithmetic costs; explicit budgets supply the remaining learning circuitry. All numbers here are projections for the proposed architecture in Appendix J.

## K.1 PUBLISHED REFERENCES AND COMPONENT BUDGETS

For a reported efficiency $s$ TOPS/W, with one MAC counted as two operations, the equivalent cost is $2000/s$ fJ/MAC. Table 28 records the precision and measurement scope of each reference. These scopes differ: a macro, its periphery, and a complete workload do not have identical energy boundaries. We transfer the reported values as alternative effective MVM costs while keeping the learning budget explicit.

Table 28: Effective MVM energy references. "Measured" refers to the cited hardware's operation, while "design" and "ideal" identify projected levels. These chips are not assumed to implement the complete PerProp learning circuit.

| Reference | fJ/MAC | Precision and scope |
|---|---|---|
| HERMES, measured (Le Gallo et al., 2023) | 204.92 | One-phase MVM at 9.76 TOPS/W, 8-bit I/O and analog weights. The paper's high-accuracy mode uses four phases. |
| Xue ReRAM, measured (Xue et al., 2021) | 167.93 | 11.91 TOPS/W at 8-bit input and weight, 14-bit output. The 195.7 TOPS/W binary mode is not used. |
| Khwa mixed CIM, measured (Khwa et al., 2025) | 48.89 | 40.91 TOPS/W workload-level efficiency for mixed integer/floating-point ResNet-20 on CIFAR-100. |
| FACIM, measured (Huang et al., 2026) | 17.09 | 117 TOPS/W with analog transmission, analog I/O and INT8 weights. The 1,880 TOPS/W macro-only figure is not used. |
| RPU, design (Gokmen & Vlasov, 2016) | 9.52 | Projected 210 TOPS/W tile and periphery with 5-bit input and 9-bit output. |
| Ideal hardware | 4.00 / 1.00 | Effective MVM budgets assuming adequate precision for the target computation. |

The 9.52 fJ design level is the one used in the main scaling figure. The 4 and 1 fJ levels explore ideal hardware with lower MVM cost. They retain all other budgets in Table 29; reducing the MVM cost does not remove feedback learning, programming, or neuron energy.

Table 29: Component budgets used throughout the projections. Costs are per listed operation, before the additional 10% routing allowance.

| Operation | Budget | Basis |
|---|---|---|
| Forward / feedback MVM | Table 28 | Effective arithmetic reference. |
| Feedback correlation product | 0.1–1 pJ | Design sensitivity; 0.1 pJ in the main table. |
| Gradient-product accumulation | 2.387 fJ | Transferred from the RPU stochastic-update projection. |
| Neuron evaluation | 16.5 pJ | Project SPICE design: 330 $\mu$W for 50 ns. |
| Signal hold / scalar operation | 1 pJ | Design budget per held value or operation. |
| Slope and credit arithmetic | 2 pJ | Design budget per neuron and example. |
| Shared reciprocal | 5 pJ | Design budget per input coordinate. |
| Random probe generation | 1 pJ | Design budget per probe coordinate. |
| Forward-weight finalization | 1 pJ | State access, scaling, momentum, and programming. |
| Feedback finalization | 1 pJ | Accumulator access, normalization, EMA, and programming. |
| Final-output ADC | 10 fJ/step | 8-bit conversion; $10 \times 2^8$ fJ per output. |
| Loss and control | 100 pJ | Per example, or per token in the Transformer projection. |
| Additional routing | 10% | Applied to the accounted analog component total. |

The gradient-product budget corresponds to $1000/419$ fJ per product from the published RPU update-cycle estimate (Gokmen & Vlasov, 2016). Transferring this budget assumes that the target accumulator provides sufficient precision. The signal, normalization, and programming entries are engineering budgets for the proposed circuits, rather than measurements of an integrated PerProp

chip. The routing factor is an additional network allowance, following the form of the AIS estimate (Wang & Achour, 2026).

### K.2 Crossbar operation counts

Consider a residual network with $D$ blocks of width $h$, 3,072 inputs and ten outputs. Its $2D + 2$ arrays contain

$$P = 3072h + 2Dh^2 + 10h$$

weights. Let $P_0 = 3072h$ be the input-projection weights, $Q$ and $R_o$ the sums of array input and output dimensions, and $H = (D + 1)h$ the number of nonlinear neurons. For batch size $b$, each update uses $bP$ forward MACs, $b(P-P_0)$ feedback MACs, and $bP$ gradient products. The first layer needs no input error. Each example evaluates every neuron nominally and at both perturbations.

There are $m = 96$ feedback probes per update and $K_0 = 20$ initialization refreshes. Amortizing initialization over $K = 12{,}000$ updates gives $r = 1 + K_0/K$. Feedback identification uses $rmP$ forward-probe MACs and $rmP$ correlation products per update, including the input projection. It also uses $rmQ$ generated probe coordinates and squares, $r(m - 1)Q$ denominator additions, $rQ$ reciprocals, $rm(Q + R_o)$ probe-response holds, and $rP$ feedback finalizations.

Write the corresponding component costs as $e_{\mathrm{MVM}}$ for a MAC, $e_\phi$ for a neuron evaluation, $e_g$ for a gradient product, $e_c$ for a correlation product, $e_h$ for a held value, and $e_s$ for a scalar operation. The remaining subscripts name the operations in Table 29. The shared training budget is

$$\begin{aligned} E_{\mathrm{shared}} = {} & b(2P - P_0)e_{\mathrm{MVM}} + 3bHe_\phi + bHe_{\mathrm{slope}} + b(Q + R_o + 2H)e_h \\ & + 2bDhe_s + bPe_g + Pe_W + be_{\mathrm{ctl}}, \end{aligned} \tag{32}$$

and feedback learning adds

$$\begin{aligned} E_{\mathrm{learnB}} = {} & rmP(e_{\mathrm{MVM}} + e_c) + rmQ(e_{\mathrm{rnd}} + e_s) + r(m - 1)Qe_s \\ & + rQe_{\mathrm{rec}} + rm(Q + R_o)e_h + rPe_B. \end{aligned} \tag{33}$$

Thus $E_{\mathrm{Hybrid}} = 1.1E_{\mathrm{shared}}$ and $E_{\mathrm{PerProp}} = 1.1(E_{\mathrm{shared}} + E_{\mathrm{learnB}})$. Inference uses only forward arrays and neurons:

$$E_{\mathrm{infer}} = 1.1\left[Pe_{\mathrm{MVM}} + He_\phi + Dhe_s + 10(2^8 e_{\mathrm{ADC}}) + e_{\mathrm{ctl}}\right]. \tag{34}$$

Unused learning circuits are assumed to be power-gated during inference. PerProp and Hybrid therefore have the same inference budget for the same forward network.

Ordinary drive and sensing are included in the effective MVM reference, so the model does not add another ADC after each array. Separate final-output conversion is charged. The model is a component budget; layout-specific leakage, utilization, data staging, and inter-chip transfer require a concrete implementation to evaluate.

### K.3 Network size, batch amortization, and digital references

The four sizes use $(D, h) = (8, 128), (16, 512), (16, 2048), (16, 8192)$, giving 0.66M, 9.97M, 140.5M, and 2.17B weights. The first is the simulated CIFAR-10 topology. The others extrapolate its architecture and operation counts. Tables 30 and 31 use batch 2,048, which amortizes each feedback refresh across more examples while holding $m = 96$ fixed.

Table 30: Projected PerProp energy per example ($\mu$J): training / inference. Training uses batch 2,048; the correlation budget is 0.1 pJ/product. Only the 0.66M network has the crossbar accuracy experiment; the other columns extrapolate the architecture.

| Effective MVM reference | 0.66M | 9.97M | 140.5M | 2.17B |
|---|---|---|---|---|
| Measured: HERMES (1 phase) | 0.300 / 0.170 | 4.91 / 2.41 | 67.0 / 32.3 | 1,025 / 492 |
| Measured: Xue ReRAM | 0.261 / 0.143 | 4.14 / 2.01 | 55.5 / 26.6 | 845 / 404 |
| Measured: Khwa mixed CIM | 0.137 / 0.0575 | 1.68 / 0.703 | 18.7 / 8.23 | 266 / 120 |
| Measured: FACIM | 0.103 / 0.0345 | 1.02 / 0.355 | 8.83 / 3.31 | 112 / 43.5 |
| Design: RPU | 0.0953 / 0.0291 | 0.862 / 0.272 | 6.48 / 2.14 | 74.7 / 25.4 |
| Ideal: 4 fJ | 0.0895 / 0.0251 | 0.747 / 0.211 | 4.78 / 1.29 | 47.9 / 12.2 |
| Ideal: 1 fJ | 0.0864 / 0.0229 | 0.685 / 0.178 | 3.85 / 0.823 | 33.3 / 5.06 |

For a digital MAC cost $e_d$, the comparison counts $b(3P - P_0)$ MAC-equivalent training operations and $P$ inference MACs. Training uses the historical GPU reference $e_d = 29.4$ pJ/MAC, derived from the GPT-3 energy and computation figures reported by Patterson et al. (2021). Inference uses $e_d = 1.4$ pJ/MAC, obtained from the H100 SXM's dense BF16 peak throughput and thermal design power (NVIDIA, n.d.). The latter is a peak-power proxy, not a measured workload trace. These

references use different digital hardware and accounting conditions; Table 32 also reports training against the H100 reference.

Table 31: Projected efficiency ratios: training / inference. Training compares with the 29.4 pJ/MAC historical digital reference; inference compares with the 1.4 pJ/MAC H100 peak-power reference. Numerators use the same network and operation budget, not matched predictive quality.

| Effective MVM reference | 0.66M | 9.97M | 140.5M | 2.17B |
|---|---|---|---|---|
| Measured: HERMES (1 phase) | 155 / 5.4 | 170 / 5.8 | 182 / 6.1 | 186 / 6.2 |
| Measured: Xue ReRAM | 178 / 6.4 | 201 / 6.9 | 220 / 7.4 | 226 / 7.5 |
| Measured: Khwa mixed CIM | 340 / 16.0 | 497 / 19.8 | 654 / 23.9 | 717 / 25.5 |
| Measured: FACIM | 449 / 26.6 | 818 / 39.4 | 1383 / 59.4 | 1711 / 69.9 |
| Design: RPU | 486 / 31.6 | 967 / 51.4 | 1883 / 91.9 | 2554 / 119.6 |
| Ideal: 4 fJ | 518 / 36.7 | 1115 / 66.1 | 2557 / 152.9 | 3988 / 248.7 |
| Ideal: 1 fJ | 536 / 40.2 | 1216 / 78.3 | 3174 / 239.1 | 5736 / 600.9 |

The ratios compare the same network and useful operation schedule. They do not compare the energy required to reach equal accuracy. The 2.17B column has no additional training-accuracy experiment. Lower MVM energy and larger batches reduce the cost per example, while neuron and learning-circuit costs remain in the total.

Table 32: Learning-circuit sensitivity for the 2.17B-weight architecture at batch 2,048. Only correlation energy varies; both digital columns refer to training.

| MVM reference | Correlation (pJ) | $\mu$J/example | Historical ratio | H100 ratio |
|---|---|---|---|---|
| Measured: FACIM | 0.1 | 111.570 | 1711.0 | 81.5 |
| Measured: FACIM | 1.0 | 212.566 | 898.1 | 42.8 |
| Design: RPU | 0.1 | 74.744 | 2554.0 | 121.6 |
| Design: RPU | 1.0 | 175.741 | 1086.2 | 51.7 |
| Ideal: 4 fJ | 0.1 | 47.873 | 3987.5 | 189.9 |
| Ideal: 4 fJ | 1.0 | 148.870 | 1282.3 | 61.1 |
| Ideal: 1 fJ | 0.1 | 33.280 | 5736.1 | 273.1 |
| Ideal: 1 fJ | 1.0 | 134.276 | 1421.7 | 67.7 |

Table 32 varies the correlation budget from 0.1 to 1 pJ/product. This is a circuit-design sensitivity, not a statistical confidence interval. Table 33 returns to batch 128, the actual batch of the crossbar accuracy experiment, and includes Hybrid alongside PerProp.

Table 33: Projected energy for the simulated 0.66M-weight topology at its training batch of 128. Ranges vary the correlation budget from 0.1 to 1 pJ/product. Training is $\mu$J/update; inference is nJ/example.

| MVM reference | PerProp training | Hybrid training | Inference |
|---|---|---|---|
| Measured: HERMES (1 phase) | 61.89–124.40 | 37.52 | 170.19 |
| Measured: Xue ReRAM | 54.53–117.04 | 32.72 | 143.47 |
| Measured: Khwa mixed CIM | 30.84–93.35 | 17.30 | 57.49 |
| Measured: FACIM | 24.52–87.03 | 13.18 | 34.52 |
| Design: RPU | 23.01–85.52 | 12.20 | 29.05 |
| Ideal: 4 fJ | 21.91–84.42 | 11.49 | 25.06 |
| Ideal: 1 fJ | 21.31–83.82 | 11.10 | 22.90 |

### K.4 Projected Transformer training energy

The energy panel uses the five model-size configurations and their achieved validation scores from Appendix F. Their operation counts are mapped to the 9.52 fJ/MAC design level, with the same support-circuit budgets as above. This is an extension of the proposed analog architecture to the recorded Transformer workloads; the validation results themselves come from digital runs.

The count includes learned dense maps and the vocabulary head; attention score, value, and error products; nominal and perturbed neuron evaluations; normalization and output-loss arithmetic; signal storage; embedding access; and parameter-state finalization. For attention, all six forward/backward matrix products are charged, and their temporary right-hand operands incur programming costs even where reuse could reduce them. The full-attention matrix total agrees with the FLOP calculation in Appendix F.

Each data-parallel rank pays its own feedback cost, including 20 initialization refreshes and one refresh after every weight update. Feedback correlations use 0.1 pJ/product in Table 34; the released data also retain the 1 pJ sensitivity. The analog total receives the additional 10% routing allowance.

Evaluation, host data loading, and explicit inter-replica communication are outside this operation budget. The digital BP energy uses its baseline FLOP estimate divided by two and multiplied by 29.4 pJ/MAC.

Table 34: Total projected training energy (kJ) for the model-size runs, using the 9.52 fJ/MAC design level in the main scaling figure. BP uses the historical digital training reference.

| Parameters (M) | Digital BP | Analog Hybrid | Analog PerProp | BP / PerProp |
|---|---|---|---|---|
| 44.302 | 253.828 | 0.405 | 0.406 | 625.2 |
| 49.612 | 652.459 | 1.220 | 1.223 | 533.5 |
| 60.234 | 1884.128 | 4.086 | 4.091 | 460.5 |
| 162.935 | 21424.389 | 31.999 | 32.116 | 667.1 |
| 407.588 | 216704.873 | 292.953 | 294.308 | 736.3 |

The resulting horizontal axis compares projected energy at the executed architecture and token budgets. It leaves the measured vertical loss values unchanged, including the remaining PerProp/BP loss gap. Energy efficiency and predictive quality are therefore visible separately in the scaling plot.