$\vec{\delta}$ Multiplexed Gradient Descent: *Perturbative Learning with Astrocytes*

Ryan O'Loughlin[1,2], Bakhrom Oripov[1], Nick Skuda[1] Noah Chongsiriwatana[3], Ian Whitehouse[3], Wolfgang Losert[3], Bradley Hayes[2], Adam McCaughan[1], Sonia Buckley[1],

[1]Physical Measurements Laboratory - National Institute of Standards and Technology (NIST), Boulder, United States
[2]Department of Computer Science - University of Colorado Boulder, Boulder, United States
[3]Department of Physics - University of Maryland (UMD), Maryland, United States
Contact: ryan.oloughlin@colorado.edu, sonia.buckley@nist.gov

***Abstract*—We present $\vec{\delta}$-Multiplexed Gradient Descent ($\vec{\delta}$-MGD), a fully neuromorphic algorithm that consistently performs alongside backpropagation with orders of magnitude fewer gradient component calculations across a number of datasets and network architectures. This performance is enabled by the finding that perturbative methods perform near-optimally when parameters are perturbed in the same direction as the true ternary gradient, and that even a noisy estimate of the gradient yields marked improvements in learning performance and convergence time. We show that local synaptic information can be used to make strong estimates of the ternary gradient using only presynaptic activation and postsynaptic bias gradients. Moreover, we find the latter can be cast into learnable class-wise representations, which we call *DeltaVecs*. According to these findings we realize a reduction of $N_{samples} \cdot N_{neurons}^2 \rightarrow N_{classes} \cdot N_{neurons}$ in terms of gradient components that must be explicitly computed per epoch. We further find that astrocyte-inspired subnetworks are able to learn DeltaVec representations in order to drive gradient-informed perturbations. Finally, we stack these methods together and show that using only neuro-inspired hardware-friendly operations we can perform alongside backpropagation in networks on the scale of millions of parameters, in DNNs, CNNs, and even transformers. Altogether, we offer an algorithm implementing novel machine learning methods and neuro-inspired operations to realize compelling neuromorphic intelligence.**

***Index Terms*—neuromorphic computing, neuromorphic algorithms, neuromorphic hardware, astrocytes, three-factor learning, Hebbian learning, ternary gradients, model-free, multiplexed gradient descent, perturbative methods, gradient dynamics**

I. Introduction

Neuromorphic computing offers a viable path toward specialized hardware for artificial intelligence by leveraging neuro-inspired physical-computing advantages for improved design of intelligent systems. A key challenge in neuromorphic computing is the development of hardware-suitable learning algorithms that are competitive with the backpropagation performance that drives nearly all learning in modern AI on traditional hardware. This work proposes a novel connection between astrocytes and perturbative methods, finding that astrocyte-like subnetworks can use locally-estimated compressed gradient representations to drive perturbations in order to realize massive performance gains.

We present *$\vec{\delta}$-MGD*, a hardware-friendly neuromorphic learning algorithm that consistently performs alongside backpropagation across a number of datasets and network architectures, including DNNs, CNNs, and transformers (preliminary), with orders of magnitude computational savings that grow with network size. $\vec{\delta}$-MGD is enabled by a sequence of novel observations and techniques that culminate in a fully neuromorphic learning scheme. These findings are enumerated below, with labels corresponding to their respective section numbers:

This work was made possible by The National Institute of Standards and Technology along with The University of Colorado Boulder.

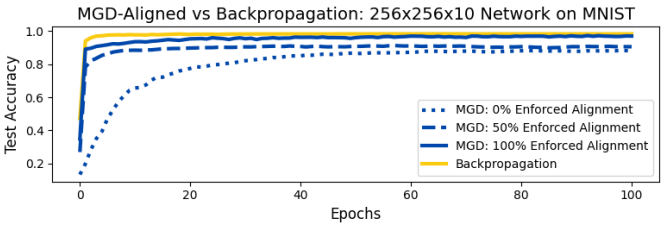

Fig. 1. *MGD with gradient-aligned perturbations:* As the percentage of perturbations that are directed into signed-agreement with the ternary gradient increases, performance and convergence time are both improved.

II **Gradient-aligned perturbations (MGD-aligned):** Perturbative methods can achieve backpropagation-like performance by aligning perturbations with a low-precision, noisy gradient estimation, as seen in Fig. 1.

III **Local gradient estimation (Hebbian MGD):** Using a Hebbian-like correlation of locally available information, a gradient estimation can be made to inform perturbation directions and implement MGD-aligned.

IV **Class-wise bias-gradient representations (DeltaVecs):** Bias gradients are found to have significant per-class structure that enables a massive reduction in the gradient search space, and offers a path to compact gradient representations with persistent viability over time.

V **Intelligent subnetworks (astrocytes):** Astrocyte-like subnetworks can serve to intelligently drive perturbations in more useful directions. In particular, they are able to efficiently learn, or sample, bias gradient representations and apply them according to local information in order to drive gradient-informed perturbations.

VI **Fully neuromorphic implementation ($\vec{\delta}$-*MGD*):** $\vec{\delta}$-MGD achieves backpropagation-like performance at a fraction of the operational cost by using astrocytes to estimate local class-wise gradient representations and drive aligned perturbations.

Sections I-V present standalone findings that together support

the final algorithm $\vec{\delta}$-MGD of Sec. VI and lead to the final results in VII. These findings mark a successful case of neuro-inspired improvements to hardware-compatible machine learning performance and are believed to be state-of-the-art perturbative performance in terms of accuracy over efficiency quotient [1]–[5].

II. Gradient-informed perturbations

We find that aligning perturbation vectors with a low-precision gradient estimate yields massive performance gains by maximizing perturbative signals measurable in the global cost. This finding is applied in the context of the multiplexed gradient descent (MGD) framework, which when directed with gradient-informed perturbations is referred to as *MGD-aligned*.

A. Perturbative Methods

Generally, perturbative methods correlate local parameter changes with a global change in cost in order to estimate the network gradient. Multiplexed gradient descent (MGD) [3] is a model-free gradient-descent framework through which perturbative methods, such as SPSA [1], [2] and finite-difference [6], can be implemented with hardware conscientiousness by introducing a trio of time constants τ_x, τ_θ, and τ_p governing the perturbation process. For example, SPSA is realized within the MGD framework when τ_θ=1 (simultaneous perturbation of all parameters) and $\tau_p = 1$ (updating parameters after every round of perturbations). Perturbations are drawn from a Bernoulli distribution where each value has an equal and independent chance of being assigned $\pm\epsilon$. In SPSA, gradients are estimated for the i^{th} parameter according to

$$\hat{\nabla}_i \leftarrow \frac{\Delta C \cdot \tilde{\theta}_i}{\epsilon^2} \approx \frac{\Delta C}{\Delta \theta_i} \approx \frac{\partial C}{\partial \theta_i} \tag{1}$$

where $\hat{\nabla}_i$ is the i^{th} component of the estimated gradient and ΔC is the measured change in global cost with respect to an unperturbed forward pass. Referring to Fig. 2 (top), we see that the true sign of parameter-wise ΔC_i is not always correctly represented by the aggregate measure ΔC. This is because SPSA cannot directly measure each contribution ΔC_i, but rather only sees their aggregate through total change in network cost ΔC. The finite difference method [6] perturbs parameters one-at-a-time to measure perfectly their individual contributions to cost ΔC_i, but pays for this estimation accuracy with required forward passes per gradient-estimate scaling with the total number of network parameters N_Θ. MGD benefits from a more practical approach whereby repeated applications of Eqn. 1 with freshly drawn perturbation vectors will strengthen the statistical correlation of each estimated component with its true gradient by way of temporal integration. Increasing integration time τ_θ up to $\tau_\theta = N_\Theta$ will recoup the same estimation accuracy as finite difference [2], but again at the cost of N_Θ scaling. A key finding of MGD, however, is that perfect gradient estimations are not required to perform gradient descent, and that hardware efficiency and machine learning performance can be balanced by moderating τ_θ [5].

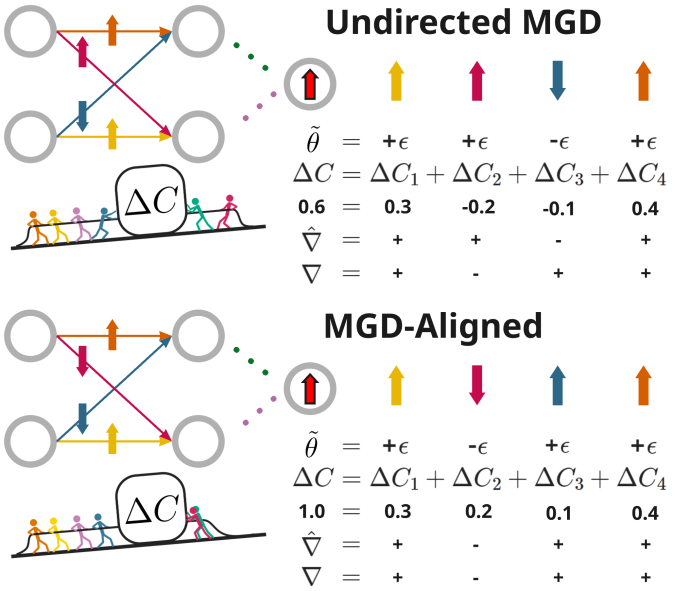

Fig. 2. (Top) **Undirected MGD:** Weights are subject to random perturbations $\tilde{\theta}$ sampled from $\pm\epsilon$. Each perturbation $\tilde{\theta}_i$ affects the cost of the network, changing it by ΔC_i. $\hat{\nabla}$ is estimated according to $\Delta C \cdot \tilde{\theta}_i/\epsilon^2 \approx \Delta C/\Delta\tilde{\theta}_i$ for each component, and we can see here that the sign of this estimate is not always in the same direction as what would have been produced if ΔC_i were known creating a mismatch between estimated gradient $\hat{\nabla}$ and true gradient ∇. We can also observe, that the estimated gradient components that have the correct sign correspond to the majority of true total gradient magnitude, which is proportional to the unknown values ΔC_i. This is why undirected MGD still performs gradient descent, even if inefficiently. Conflicting contributions to cost are analogized to the disorderly stick-figure efforts depicted alongside the worked example. (Bottom) **MGD-Aligned:** Here we observe a straightforward remedy to the self-cancellation of undirected-MGD. When perturbations are directed such that a common sign will be produced by all ΔC_i components, *all* of their respective impacts on the network are accounted for in the ΔC measure, whose magnitude is thus increased. Each parameter is still updated with correct signage according to Equation 1 and we see all estimated components align with the true gradient.

TABLE I
Gradient calculation methods, estimation strength, scaling, and hardware viability.

∇ Method	$\frac{\partial C}{\partial \theta_i} \propto \frac{\Delta C}{\Delta \theta_i}$	Scaling	HW-friendly
Backprop	Perfect	$\mathcal{O}(bp)$	✗
Finite Difference	Very Strong	$\mathcal{O}(N_\Theta \cdot fp)$	✓
MGD($\tau = 1$) (SPSA)	Weak	$\mathcal{O}(fp)$	✓
MGD($\tau = T$)	Dependent	$\mathcal{O}(T \cdot fp)$	✓
MGD($\tau = N_\Theta$)	Very Strong	$\mathcal{O}(N_\Theta \cdot fp)$	✓
MGD-Aligned	Very Strong	$\mathcal{O}(fp)$	✓

bp = one backward pass, fp = one forward pass

The performance/efficiency tension is thus as follows (as listed in Table I): If $\tau_\theta = N_\Theta$, gradient estimations are very strong, but scaling the network will punish convergence time. On the other hand, if $\tau\theta = 1$, gradient estimates are poor, but there is no punishment for scaling the number of parameters in a network. In general, we would like to maximize gradient estimate accuracy and minimize integration time. A long standing open question of perturbative methods is whether the performance/efficiency ratio could be improved

by more selective perturbation vectors. To answer this, we first investigate where misalignment is accumulating according to random perturbations.

B. Washout

If a parameter is perturbed alone, the change in cost might be measured as being in a certain direction, but when that same perturbation is obfuscated by the result of all parameters being perturbed simultaneously, the change in cost may be measured as being in the *opposite* direction. Eqn. 1 approximates a measurement of the true gradient component $\frac{\Delta C_i}{\Delta \theta_i}$ with the efficiently measurable value $\frac{\Delta C}{\Delta \theta_i}$. This approximation simplifies the task of measuring perturbation-to-cost correlations one-by-one to a single measurement of how all perturbations associate collectively with a change in global cost. Each i^{th} parameter is affected by its own perturbation $\tilde{\theta}_i$, which can be then be related directly to the total ΔC. Again referring to Fig. 2 (top), we see that the true parameter-wise contribution to cost is not always represented in the measured change in global cost. For example, the dark red weight is subject to a positive perturbation, and a positive change in global cost is measured. This drives our Eqn. 1 estimate to assume that making this parameter bigger is increasing the cost of the network, suggesting a *positive gradient* and *negative update*. However, we see that in reality this parameter is associated with a negative contribution to the cost, meaning that increasing this parameter value would further minimize loss. This should ideally result in a *negative gradient* estimate and *positive update*. We therefore find that undirected MGD, which employs random Bernoulli perturbations, necessarily suffers from some percentage of parameters receiving erroneous updates. This is caused by their contributions to the global cost being canceled (*washed out*) by conflicting contributions from other parameters.

This characterization of the perturbation-cost relationship lends itself to a number of useful insights. First, in undirected MGD each parameter might receive a good, bad, or neutral update according to the sign agreement of ΔC and ΔC_i. The neutral case is that of weight diffusion [4], whereby updates are made to a parameter despite it having no correlation with the cost. Second, because local correlation is estimated according to Eqn. 1, and the measured change in cost is determined by $\Delta C = \sum_i^{N_\Theta} \Delta C_i$, the sign of ΔC will always represent the group of parameters that correlate with the majority impact on cost. Realize that each ΔC_i is determined by its corresponding $\tilde{\theta}_i$. When the $sign(\Delta C_i) = sign(\Delta C)$ then the estimate is in the correct direction. Because per-parameter perturbations are what drive ΔC, the group of parameters with the strongest collective correlation to the cost will always determine the sign of ΔC and receive good updates. Equivalently, this implies that the angle between the estimated gradient and the true gradient should always be acute $\cos(\hat{\nabla}, \nabla) < 90°$ by virtue of the majority of gradient magnitude being represented by correctly-aligned estimates. This new insight is empirically supported in [5] and preempts the basis for deriving perturbation vectors that will maximize performance by minimizing washout and improving gradient estimations.

C. MGD Aligned

By inspection of Fig. 2, we can see that whatever the true gradient for a given component is, a set of perturbations exists such that a common sign is produced among all components ΔC_i, thus maximizing perturbative visibility in the cost, and guiding all component updates in the right direction. Enforcing this perturbative signal maximization is exactly equivalent to directing all perturbations into signed-agreement with the true ternary gradient, such that the correct relationships to their true gradient directions will be realized across a common direction for the measured change in cost. The same result is seen when perturbations are all directed together in the anti-direction of the ternary gradient, thus flipping ΔC. The key detail is that all perturbations should have the *same* correlation with the true gradient. To confirm this intuition, we turn to

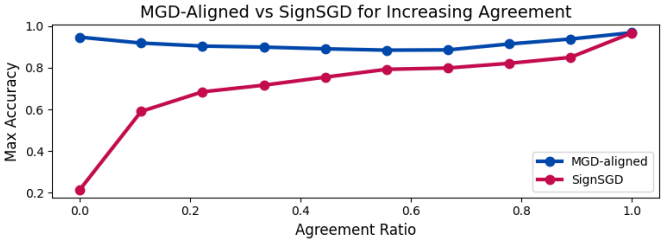

Fig. 3. MGD vs SignSGD for different agreement ratios. SignSGD applies $\tilde{\theta}$ directly as a parameter update, whereas MGD utilizes $\tilde{\theta}$ as a perturbation vector. On the far left algorithms receive a random Bernoilli vector of $\pm\epsilon$ values for $\tilde{\theta}$. On the far right, $\tilde{\theta} = \epsilon \cdot sign(\nabla)$ such that $\Lambda = 1$. Each data point is the best performance of a sweep over ten learning rates, as the effective learning rate may be shifting for different $\tilde{\theta}$ distributions, which is likely the cause of the imperfect bowing in MGD performance.

empirical measurements. We define an *agreement measure* Λ that computes the ratio of components in a vector A that are in ternary-signed agreement with those in another vector B

$$\Lambda = \frac{\sum_{i=0}^{N}[sign(A) = sign(B)]}{N}, \tag{2}$$

where N is the total number of components. Fig. 1 compares three different perturbation vector conditions against backpropagation, where 0%, 50%, and 100% of perturbation-vector components are enforced to agree with the true ternary gradient. When 100% of components are subject to this condition, $\Lambda = 1$. The effect of Λ on performance in terms of accuracy and convergence speed is unambiguous: the higher the Λ value, the better the accuracy and the faster the convergence. For $\Lambda = 1$ MGD performance adheres closely to that of backpropagation. We define *MGD-aligned* to be the case where $\Lambda(\tilde{\theta}, \nabla) \approx 1$, in contrast to *Undirected MGD* whose Bernoulli perturbations only result in $\cos(\hat{\nabla}, \nabla) < 90°$.

We find that MGD still performs gradient descent anywhere on the agreement spectrum from undirected to aligned. This shows that noisy estimations of the ternary gradient suffice for performance improvements. While it may be tempting to apply the ternary gradient directly toward learning, as is the

case with SignSGD [7], we see in Fig. 3 that this approach is considerably less robust to noise. It should be stated that the motivation for SignSGD is to efficiently communicate gradient information across GPUs, rather than to realize neuromorphic learning on specialized hardware. The reason MGD is so much more noise tolerant is that updates are made proportional to ΔC, which is itself an empirical measure of how strong the ternary gradient estimate is correlated with the true gradient of the network. However, as already seen in Fig. 1, noisy perturbation vectors still cost in terms of performance and convergence speed. We are therefore motivated to investigate neuromorphic means for making strong ternary gradient estimates online and enable the scaling advantages outlined in Table I.

III. Hebbian MGD: Deriving Ternary Gradient from Local Information

To access the performance gains of MGD-aligned, an estimate of the ternary gradient is required. Computing the gradient analytically, ternary or otherwise, is infeasible in dedicated hardware and we therefore propose a gradient-estimation method that is suitable for *in-situ* hardware implementation.

A. Estimation by Local Information

Perturbative methods can be implemented according to neuromorphic operations, but suffer with parameter scaling. This is unfavorable given that the number of learnable parameters (weights, biases, etc) N_Θ in a network scales roughly squared with the number of of neurons in a network $N_\Theta \propto N_n^2$. However, upon closer inspection of the problem space, as depicted in Fig. 4, we see that for any given weight parameter W_{ij} connecting neuron n_i with n_j, the partial derivative with respect to the cost function is given by the presynaptic activation a_i and the postsynaptic error term δ_j. Presynaptic activation is simply the value computed during a forward pass by a weight's upstream neuron and is available locally at the site of derivation. The postsynaptic delta is the bias gradient at the downstream neuron $\delta_j = \frac{\partial C}{\partial b_j}$, which can equivalently be thought of as the gradient at the postsynaptic neuron.

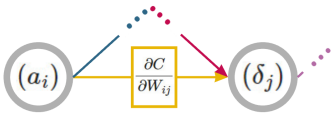

Fig. 4. Any given network weight connects exactly two neurons $n_i \rightarrow n_j$. Its local gradient $\frac{\partial C}{\partial W_{ij}}$ is given by the activation value a_i of the *upstream* neuron n_i, and the delta δ_j of the *downstream* neuron n_j.

The consequence of this relation is that the gradient need only be computed for every *neuron* N_n in a network rather than for every *element* N_Θ, where $N_\Theta >> N_n$. The reduction of $\mathbb{R}^{N_\Theta} \rightarrow \mathbb{R}^{N_n}$ immediately enables improved scaling for perturbative gradient-estimation methods. This same insight underpins the *node perturbation* learning algorithm [1], [4], [8], [9], where finite difference is used to compute only bias gradients, which are applied along with presynaptic activations to perform single-layer backpropagation. Similar to SignSGD, however, node-perturbation is less noise robust than MGD as it expands on any error in gradient estimation by propagating it to other elements. MGD-aligned instead weights updates according to a true measurement of the gradient estimate's strength, making it robust to noise as already demonstrated in Fig. 3, though we still may opt to employ single-layer backpropagation for the *final* layer only, where it is an exact, local computation. For the remaining layers, only the signs of a_i and δ_j, reducing the resolution of the task from full-precision to ternary values according to

$$sign\left(\frac{\partial C}{\partial W_{ij}}\right) = sign(a_i) \cdot sign(\delta_j). \tag{3}$$

This approach is attractive for specialized hardware implementation because activation values are available locally, only N_n *ternary* gradient elements need be estimated, and only a local ternary multiplication is required for each weight in the network, all of which are desirable qualities for dedicated circuit operations.

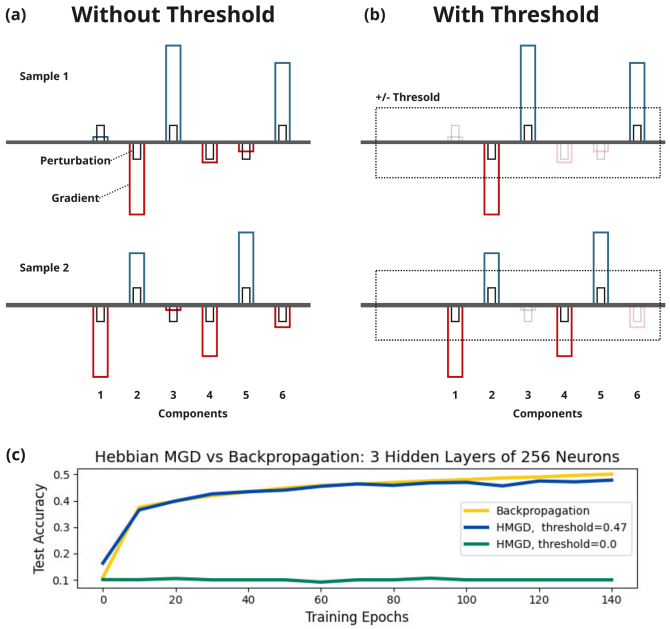

Fig. 5. **(a) Without Threshold:** Perturbations are made for every component. We see that while the signed direction may be in agreement with the gradient for each component, the difference in magnitude can widely vary, as seen in component 1. Thus, the inter-sample per-component variance is high, which is problematic because perturbation magnitude remains constant. For roughly equivalent ΔC across samples, these parameters will receive the same size updates in *opposite directions*. This is possible because *intra*-sample component-wise variance can also be high, resulting in magnitudes of ΔC that are driven by the strongest components, misrepresenting the influence weaker ones. **(b) With Threshold:** We see that by setting some $\pm$ threshold value, small components will neither interfere with more significant gradients across samples, nor be subject to large updates within a sample according to other components. **(c) Hebbian MGD:** We see thresholding is necessary for the network to learn on a per-sample basis *and* that it is sufficient to keep pace with full-precision backpropagation on the CIFAR-10 dataset.

B. Thresholding

The measurement of presynaptic activation a_i is accompanied by a caveat not previously addressed in our above implementation of MGD-Aligned. Presynaptic activations are a *per-sample* quantity. Moreover, dedicated in-memory hardware

generally favors sample serialization over batching. Therefore, we must estimate the ternary gradient on a per-sample basis. A nontrivial issue arises in this regime whereby batch-averaging would have to occur over ternarized gradients rather than full-precision gradients. However, $sign(\bar{\nabla}) \neq \overline{sign(\nabla)}$. This is because a ternary summation of gradients across components neglects inter-sample per-component magnitude variance, as visualized in Fig. 5 (a). Ostensibly, using batch sizes of 1 should circumvent this issue, but in fact the effect pervades to per-sample updates because MGD and MGD-aligned apply uniform magnitude updates across all components as determined by the ΔC measured for that gradient estimation. Thus, a component might be very large in the positive direction for one sample, and very small in the opposite direction for another sample, but still be subject to equivalent size updates for similar values of ΔC, whose total value may be driven by other components entirely. The consequence to performance is shown in Fig. 5 (c).

The solution to this issue proves to be straightforward and involves only one hardware-friendly operation. By thresholding every presynaptic activation, we can approximately capture component-wise variance in our gradient estimation, preserving only the most significant components in each per-sample estimate. By way of Equation 3, replacing a presynaptic activation with zero results in a zero-value estimate of the ternary gradient. In the MGD-aligned context, this means no perturbation is made at that component and thus no update is made. The result is that MGD-aligned is now robust to multiplicative variance caused by conflicting gradient signs across samples in addition to its intrinsic robustness to additive noise . See Fig. 5 (b) for a visualization. In addition to better tailoring sample-wise gradient estimations to the larger learning objective, this method has the added benefit of increasing the signal of the remaining parameter perturbations. Selecting an appropriate threshold is simply a question of hyperparameter tuning. For a hyperbolic tangent function, we empirically find that some value marginally less than 0.5 tends to be effective, as shown in Fig. 5 (c).

C. Hebbian MGD

In summary, we find that pre-and-postsynaptic information can be used to estimate the ternary gradient in a way that relates local activity with global reward signals and circumvents the weight transport problem of credit assignment [10] in network learning. We therefore make the loose analogy to *three-factor Hebbian learning* [11], [12], and refer to this method as *Hebbian-MGD*, implemented as follows: (1) Threshold the presynaptic activation according to its absolute value, and then ternarize it. (2) Attain a ternarized bias gradient estimation by finite difference or by the novel means presented later in this work. (3) Compute the estimated sign of network weights accordingly. (4) Use this signed gradient to define the perturbation vector on a per-sample basis. (5) Proceed with MGD-aligned learning. Fig. 5 (c) demonstrates the efficacy of this method in terms of machine learning performance with respect to the backpropagation gold standard, but unlike backpropagation, Hebbian MGD is a viable choice for neuromorphic hardware.

Having shown that MGD-Aligned performance is applicable to the entirely neuromorphic means of Hebbian MGD, we now turn our attention to further reductions in the problem space of bias gradient estimations.

IV. DeltaVecs: Spatio-Temporal Stability in Class-Wise Gradients

We now present DeltaVecs, a new approach for representing bias gradients according to intrinsic gradient features, allowing for a reduction of the gradient estimation problem space by 10^6 elements for a network of about a million parameters, and by larger factors for growing network size. These findings directly support the neuromorphic algorithm stack being presented in this paper, but may also apply to advantages in gradient computation for machine learning at large.

The total number of gradient component computations involved with training a neural network is given by

$$\mathcal{O}(N_\Theta \times N_{samples} \times N_{epochs}),$$

where N_Θ is the total number of learnable network parameters, N_{epochs} is the number of epochs required to train the network, $N_{samples}$ is the number of training samples in the target dataset, and $\mathcal{O}(\cdot)$ gives the total number of gradient computations required to learn the data. We quickly see these numbers are impractically large. Even the minimal MNIST dataset has 60,000 samples and might train over ten to a hundred epochs depending on learning rate. For a small network with only two hidden layers of 128 neurons each, trained on MNIST for only ten epochs, there are *70,969,200,000* gradient component calculations to perform. Hebbian MGD already reduces this space by a factor of N_n/N_Θ because weight derivatives can be attained implicitly from bias gradients. However, the $N_{samples}$ and N_{epochs} factors remain a nontrivial expense.

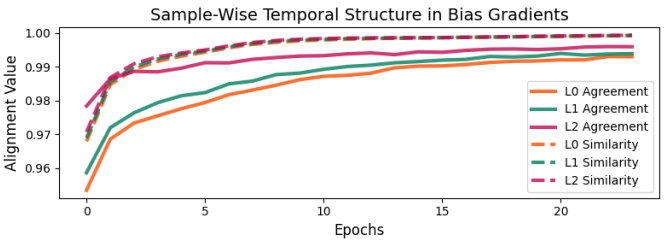

Fig. 6. Ternary agreement (solid lines) and full-precision cosine similarities (dashed lines) are measured for bias gradients from one epoch to the next during a convergent training procedure on MNIST.

Fortunately, perturbative methods are robust to noisy conditions, as already demonstrated in Sec. II. Rather than compute the gradient for every component, sample, and epoch, why not learn suitable intermediate representations? If gradients have learnable features, perhaps there can be shortcuts to calculating them analytically. There is precedence for feature analysis of neural network gradients themselves [13]–[17], but because noisy gradients are not generally satisfactory, as we have already seen with SignSGD, these sorts of efforts have

not yet been effectively directed toward gradient computation shortcuts. With respect to MGD-requirements, we assess gradient representation quality in time and space by utilization of our agreement measure defined in Equation 2 for ternary vectors, and cosine similarity for full-precision.

A. Sample-Wise Temporal Stability

The first finding we present is that the gradient vector for any given sample does not change drastically over time for a well-conditioned dataset. This is true both of the ternary and full-precision gradients, as seen in Fig. 6. While this does allude to the convexity of the loss landscape with respect to that sample, this measure is not descriptive of the gradient with respect to the entire dataset. Given this finding, we may already implement another compression to the gradient estimation problem by a factor of N_{T_δ}/N_{epoch}, where N_{T_δ} is the number of times the representation of a bias gradient might need to be updated, and $N_{T_\delta} < N_{epoch}$. The assumption is that these vectors might be sufficiently stable over time such that they need not be computed every epoch.

B. Class-Wise Stability

We see in the left quadrants of Fig. 7 that there is no apparent ternary or full-precision similarity across samples in a batch. However, a surprising richness of features becomes salient when these same bias-gradient vectors are organized according to class. Class-common features are cleanly distinguishable even by visual inspection. We can also see the strongest gradient components in the full precision plot (dark purple and bright yellow), tend to align most with these class-wise representations, and should therefore be well-captured in approximations. This offers a massive potential for gradient representation reduction by a factor of $N_{classes}/N_{samples}$. This finding, along with work such as [14]–[16], suggests that gradient dynamics are largely driven by the structure of the objective function, which requires the minimization of error to one-hot-encoded class labels. While nuanced input data features related to fundamental differences in class are essential for learning [18], these are quickly mapped to more stable gradient targets (formed by gradient directions that maximize class-wise alignment [14]) to guide separable representations of class. This sentiment may also relate to feedback alignment literature [19], which finds that even arbitrary mappings of cost to gradient estimations can produce network learning, as well as recent work on learned feedback mappings [20].

C. Class-Wise Temporal Stability

Finally, we find that the aforementioned temporal stability of bias gradients applies directly to class-wise representations, measuring $\Lambda = 0.92$ between first and last epoch of a convergent network trained on MNIST. Together with the previous structure-enable gradient compressions, we arrive at the final reduction

$$\mathcal{O}(N_\Theta \times N_{samples} \times N_{epochs}) \rightarrow \mathcal{O}(N_n \times N_{classes} \times N_{T_\delta}), \tag{4}$$

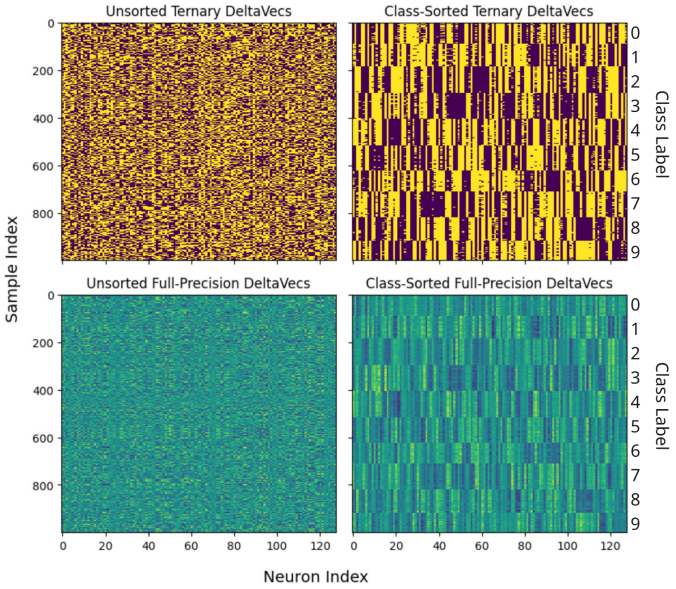

Fig. 7. (Top Left) Unsorted ternary bias gradients for a batch of training examples. Rows correspond to a sample, columns to a gradient component. (Bottom left) Unsorted full-precision bias gradients. (Top Right) Class-sorted ternary bias gradients. Strong per-class structure is now salient. (Bottom Right) Class-sorted full-precision bias gradients. Class-wise structure is also salient in the **magnitude** of components, implying strong gradient correlation across samples for a given class.

which for our sample MNIST network is

$$N_\nabla = 70,969,200,000 \rightarrow N_{\nabla_\delta} = 319,200$$

corresponding to $222,335\times$ *fewer gradient component calculations* N_∇.

D. DeltaVecs

We therefore introduce *DeltaVecs*, temporally robust class-wise representations of post-synaptic deltas (bias gradients) as an asset to efficient gradient computation methods. Equipped with massive savings in gradient estimation by way of DeltaVecs and Hebbian MGD, we next propose a way to learn, store, and produce DeltaVecs by neuromorphic means.

V. Astrocytes: Intelligent Perturbation-Generating Subnetworks

We present two findings, (1) that subnetworks offer a viable method for learning, storing, and estimating DeltaVecs by simple means, and (2) that this configuration maps well onto known and proposed astrocytic operations in the brain. These two findings complete an end-to-end stack that can guide perturbative learning to be orders of magnitude more efficient in a fully neuromorphic way.

Astrocytes are known to modulate synaptic excitability as a function of network activity and may participate in learning mechanics such as three-factor learning [21]–[24]. Moreover, a single astrocyte may connect with thousands, or even millions, of synapses [25], [26]. Similarly, the learning approaches described thus far in this work aim to drive perturbations as a function of network activity and class context in order to incite Hebbian-like three-factor learning. While there are

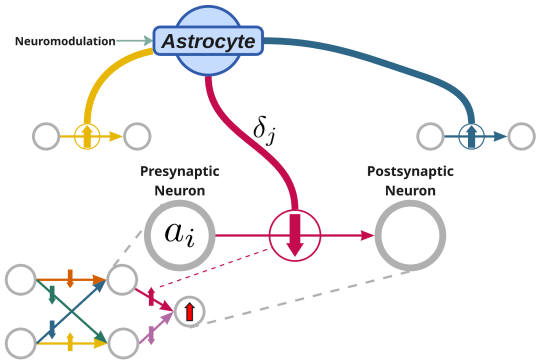

Fig. 8. **Astrocytic Model for Driving Perturbations:** A mock-up for potential astrocytic relationship with perturbative learning. An astrocyte may connect to multiple synapses and be responsible for modulating their synaptic excitability (perturbing their weights). They furthermore may be involved with perturbing in favorable directions according to locally available pre-and-postsynaptic information, and even DeltaVec-like gradient information.

myriad parallels to draw, we outline here only one of many such possible astromorphic computing paradigms.

Fig. 8 depicts a proposed model whereby an astrocyte is responsible for parameter perturbations, driven according to local activation and estimated deltas. In a multi-astrocyte scheme, each astrocyte might be responsible for its own class. Indeed, astrocytes have been shown to have context-awareness even when unsupervised [27], [28] and respond to the dynamical nature of stimulus [21]. The subset of synapses to which a class-specific astrocyte is connect could be related to the same subset of nonzero values produced by our aformentioned thresholding mechanism. A fact that supports this suggestion is that astrocytes have been shown to make local adjustments to their synaptic groups, moving a connection from one synapse to another as needed. In a supervised setting, neuromodulation might aid with class awareness, as well as gradient estimation.

A more sophisticated vision for astromorphic computation is where the onus of learning to estimate DeltaVecs is also an astrocytic responsibility. Due to the strong class-wise correlations of DeltaVecs, this should not prove inconceivable. The task can be as simple as each astrocyte sampling its corresponding class-DeltaVec every τ_δ iterations, and storing them in its subnetwork weights, which perform the linear mapping $\mathcal{A} : y \to \vec{\delta}$. These DeltaVecs could be sparsely generated only every τ_δ samples by perturbative learning. Another version of this implementation could be that the astrocytes participate in online representation forming of DeltaVecs, with only sparse training examples as determined by τ_δ. This hypothesis is tested and it is found that subnetworks are able to learn DeltaVecs and estimate them online, achieving $\Lambda > 90\%$ across all layers. Finally, astrocytes could potentially perform a second order optimization by learning to estimate gradient dynamics. While all of these possibilities are compelling, we show in the next section that the *simplest* proposed implementation (periodic DeltaVec-sampling) is able to realize superior neuromorphic performance.

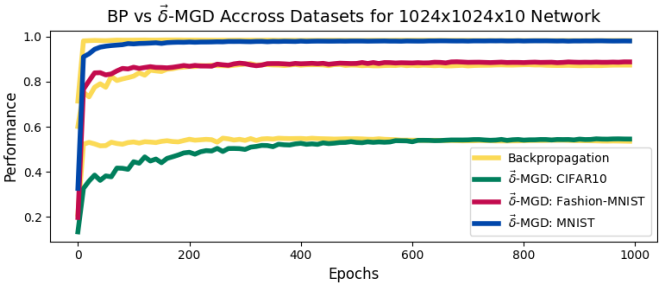

Fig. 9. We find that $\vec{\delta}$-MGD converges to backpropagation performance on all three datasets, with smooth and stable learning. While backpropagation reaches similar performance more quickly, it is requiring thousands of times more gradient component computations, which would not be implementable in neuromorphic hardware. $\vec{\delta}$-MGD uses only sparse hardware-friendly gradient estimation methods for the same performance.

VI. $\vec{\delta}$-MGD: A Neuromorphic Learning Algorithm

Finally, we arrive at a fully neuromorphic algorithm, and we find it reliably performs within 1% of backpropagation with $> 10^5$ reduction factors in N_∇ gradient component computations using only simple hardware-plausible operations across a number of architectures. We call this method $\vec{\delta}$-MGD, formally described in Alg. 1. In addition to incorporating DeltaVecs, there are few changes to previous MGD variants. Firstly, full precision gradient estimates are permitted as perturbation vectors because they are likely to estimate the sign correctly, thereby adding better-than-random (or uniform) magnitude information and offering an alternative to thresholding for the treatment of component-wise variance. This also implies that when a new DeltaVec is sampled, the update for that sample will be equivalent to backpropagation. Second, the final update is weighted by the cost C_0, according to coefficient and hyperparameter γ, to better represent inter-sample variance in gradient magnitude. The added complexity of these additions is negligible, but if hardware constraints become an issue, activation-thresholded ternary gradient es-

Algorithm 1 The $\vec{\delta}$-MGD Algorithm

Initialize network parameters Θ
for i in $N_{samples} \cdot N_{epochs}$ iterations **do**
  Input new training example x, y
  if i mod $\tau_\delta = 0$ **then**
    for class in $N_{classes}$ **do**
      Compute $\vec{\delta}$
      Update representation of $\vec{\delta}_{class}$
    end for
  end if
  Compute forward pass
  Store activations a locally, and cost C_0 globally
  Estimate gradient: $\hat{\nabla}_i \leftarrow a_i \cdot \vec{\delta}_i$
  Define $\tilde{\theta}$ according to $\hat{\nabla}_i$ and store locally
  Perturb all network parameters Θ according to $\tilde{\theta}$
  Perform forward pass
  Measure change in global cost: $\Delta C = \tilde{C} - C_0$
  Update parameters: $\Theta_i \leftarrow \eta \Delta C \cdot \tilde{\theta}_i \cdot \gamma C_0$
end for

timations work nearly as well to drive perturbations. The performance of $\vec{\delta}$-MGD is the best of any in the MGD family of algorithms, is entirely neuromorphic, and closely adheres to backpropagation, as seen in Fig. 9 and enumerated in terms of performance and savings in Sec. VII.

VII. Results

See Table II for an overview of performance results. We find $\vec{\delta}$-MGD is poised to cope with millions of parameters effectively with orders of magnitude computational savings. It is worth noting that these savings may be even more dramatic depending on the hardware-specific expense of forward versus backward passes. Efficiencies outlined by our N_∇ calculations could be considerably compounded by the speed and energy efficiencies of dedicated hardware. While listed savings are exactly true with respect to the factor fewer gradient component calculations required by $\vec{\delta}$-MGD, a complete comparison of total speed, efficiency, and performance, whether with respect to backpropagation or neuromorphic means, can only be fully understood on a per-hardware basis and as a function of implementation optimization. We expect $\vec{\delta}$-MGD to benefit from competitive hardware implementations. Furthermore, these

TABLE II
$\vec{\delta}$-MGD vs Backpropagation, Performance and Efficiency

Network	Dataset	BP	$\vec{\delta}$-MGD	% BP	×Savings
Dense Net 128x128 N_Θ=411,146	MNIST	97.9%	97.2%	**99.3%**	$\mathbf{2 \cdot 10^5}$
	Fashion	87.9%	87.2%	**99.2%**	$\mathbf{2 \cdot 10^5}$
	CIFAR	51.9%	51.2%	**98.7%**	$\mathbf{8 \cdot 10^5}$
Dense Net 1024x1024 N_Θ=855,050	MNIST	98.6%	98.2%	**99.6%**	$\mathbf{5 \cdot 10^5}$
	Fashion	88.0%	88.8%	**100.9%**	$\mathbf{5 \cdot 10^5}$
	CIFAR	55.2%	54.8%	**98.0%**	$\mathbf{1 \cdot 10^6}$
CNN 6C+4D $N_\Theta \approx 10^6$	MNIST	99.4%	99.2%[a]	**99.8%**	–
	Fashion	92.3%	90.3%[a]	**97.8%**	–
	CIFAR	74.0%	70.9%[a]	**95.8%**	–
Transformer NanoGPT $N_\Theta \approx 10^8$	Shake-speare	val loss 3.0	val loss 3.[b]	**95.0%**	–

[a]For implementation simplicity, Hebbian MGD used here.

[b]Reduced training set size, only projection matrices trained w/ $\vec{\delta}$-MGD, remainder w/ backpropagation.

mechanisms map to more sophisticated network architectures such as CNNs and transformers. For CNNs, DeltaVecs are computed in the convolutional layers for kernel weights rather than biases because they are less numerous, and then applied toward Hebbian MGD. Transformer results are with respect to $\vec{\delta}$-MGD training projection matrices in NanoGPT [29] on the Shakespeare next token prediction task [30]. The remainder of parameters are trained with backpropagation. We find that all but the the K-cache parameters have stable DeltaVecs, but that per-sample implementation is sufficiently complex to save for future work, though these results encourage the potential for energy savings in large language models.

VIII. Methods

All simulations are implemented on GPUs using the JAX Python package [31]. A functional repository and code to produce all plots can be found at: https://github.com/ryangitsit/delta-mgd. Anytime two algorithms are compared, the best of their performances over a relevant hyperparameter sweep are presented. Adam optimizer and decay are used for final results only.

IX. Discussion: Gradient Dynamics, Astrocytic Learning, and Hardware Implementations

We began this work with the empirical finding that a perturbation is improved by increasing sign-agreement with the true ternary gradient, thereby founding MGD-aligned. We then determined a hardware plausible method for computing the ternary gradient with locally available information using Hebbian-MGD. Next, by introducing DeltaVec class-wise representations of postsynaptic bias gradients, we show the problem space of ascertaining the postsynaptic information required by Hebbian MGD is reduced by several orders of magnitude, and we then show this is a task well-suited for astrocyte-like subnetworks. Finally, we observe that this stack of hardware-viable neuro-inspired mechanisms achieves backpropagation-like performance at a minuscule fraction of required gradient-component calculations for dense networks, CNNs, and transformers. These findings realize a number of implications across neuroscience, physical hardware, and machine learning.

Neuroscience: Astrocytes and perturbative methods may be strongly related in terms of their phenomenology and computational role in learning. Future work ought to focus on more tightly coupling known biological dynamics with perturbative methods.

Hardware: Global broadcast (reward signal), local perturbations, local memory, minimal finite difference, Hebbian-like mechanisms, and subnetworks are all plausible in neuromorphic hardware. Mapping this suite of operations should be investigated in known neuromorphic hardware [32]–[36], as well as explored in emerging systems that could opt to tailor physical structures to this modest recipe of operations.

Machine learning: Gradient dynamics are well-studied, but applying DeltaVecs toward short-cutting gradient computations is a novel finding. Our preliminary results on CNNs and transformers ought to be extended to pure $\vec{\delta}$-MGD implementations and on larger scales. Second-order optimization methods, by which gradient dynamics are learned early in training and utilized thereafter for continued network training, also present an area offering of potentially massive computational savings. The viability of learning gradient dynamics is supported by Fig. 6, which suggests smooth bias gradient Hessians.

In conclusion, we have presented a complete neuromorphic algorithm that is neuro-inspired, hardware-plausible, and machine learning performant. This work offers a number of further investigations to pursue, many of which may interest a diversity of academic and industry communities.

ACKNOWLEDGMENT

This work was made possible by the institutional support from the National Institute of Standards and Technology and the University of Colorado Boulder.

DISCLAIMER

Commercial hardware are mentioned to contextualize algorithm applicability, but this does not imply endorsement of any product or service by NIST.

REFERENCES

[1] Amir Dembo and Thomas Kailath. Model-free distributed learning. *IEEE Transactions on Neural Networks*, 1(1):58–70, 1990.

[2] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[3] Adam N McCaughan et al. Multiplexed gradient descent: Fast online training of modern datasets on hardware neural networks without backpropagation. *APL Machine Learning*, 1(2), 2023.

[4] Paul Züge, Christian Klos, and Raoul-Martin Memmesheimer. Weight versus node perturbation learning in temporally extended tasks: Weight perturbation often performs similarly or better. *Physical Review X*, 13(2):021006, 2023.

[5] Bakhrom G Oripov, Andrew Dienstfrey, Adam N McCaughan, and Sonia M Buckley. Scaling of hardware-compatible perturbative training algorithms. *arXiv preprint arXiv:2501.15403*, 2025.

[6] Lewis Fry Richardson. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 210(459-470):307–357, 1911.

[7] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International conference on machine learning*, pages 560–569. PMLR, 2018.

[8] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. *Advances in neural information processing systems*, 5, 1992.

[9] Sander Dalm, Marcel van Gerven, and Nasir Ahmad. Effective learning with node perturbation in multi-layer neural networks. *arXiv preprint arXiv:2310.00965*, 2023.

[10] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 2007.

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

[12] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016.

[13] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019.

[14] Satrajit Chatterjee. Coherent gradients: An approach to understanding generalization in gradient descent-based optimization. *arXiv preprint arXiv:2002.10657*, 2020.

[15] Satrajit Chatterjee and Piotr Zielinski. On the generalization mystery in deep learning. *arXiv preprint arXiv:2203.10036*, 2022.

[16] Vardan Papyan, XY Han, and David L Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

[17] Yanzhou Pan, Huawei Lin, Yide Ran, Jiamin Chen, Xiaodong Yu, Weijie Zhao, Denghui Zhang, and Zhaozhuo Xu. Alinfik: Learning to approximate linearized future influence kernel for scalable third-party llm data valuation. *arXiv preprint arXiv:2503.01052*, 2025.

[18] Guodong Zhang, James Martens, and Roger B Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[19] 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.

[20] Brian S Robinson, Isaac Western, Erik C Johnson, and Matthew J Roos. Optimizing generalized feedback paths for credit assignment. In *Proceedings of the International Conference on Neuromorphic Systems*, pages 135–140, 2025.

[21] Kate M O'Neill, Emanuela Saracino, Barbara Barile, Nicholas J Mennona, Maria Grazia Mola, Spandan Pathak, Tamara Posati, Roberto Zamboni, Grazia P Nicchia, Valentina Benfenati, et al. Decoding natural astrocyte rhythms: dynamic actin waves result from environmental sensing by primary rodent astrocytes. *Advanced Biology*, 7(6):2200269, 2023.

[22] Min Wu, Xudong Wu, and Pietro De Camilli. Calcium oscillations-coupled conversion of actin travelling waves to standing oscillations. *Proceedings of the National Academy of Sciences*, 110(4):1339–1344, 2013.

[23] Gertrudis Perea, Marta Navarrete, and Alfonso Araque. Tripartite synapses: astrocytes process and control synaptic information. *Trends in neurosciences*, 32(8):421–431, 2009.

[24] Alfonso Araque, Vladimir Parpura, Rita P Sanzgiri, and Philip G Haydon. Tripartite synapses: glia, the unacknowledged partner. *Trends in neurosciences*, 22(5):208–215, 1999.

[25] Nancy Ann Oberheim, Takahiro Takano, Xiaoning Han, Wei He, Jane HC Lin, Fushun Wang, Qiwu Xu, Jeffrey D Wyatt, Webster Pilcher, Jeffrey G Ojemann, et al. Uniquely hominid features of adult human astrocytes. *Journal of Neuroscience*, 29(10):3276–3287, 2009.

[26] Eric A Bushong, Maryann E Martone, Ying Z Jones, and Mark H Ellisman. Protoplasmic astrocytes in ca1 stratum radiatum occupy separate anatomical domains. *Journal of Neuroscience*, 22(1):183–192, 2002.

[27] Justin Lines, Eduardo D Martin, Paulo Kofuji, Juan Aguilar, and Alfonso Araque. Astrocytes modulate sensory-evoked neuronal network activity. *Nature communications*, 11(1):3689, 2020.

[28] Rune Nguyen Rasmussen, Antonis Asiminas, Eva Maria Meier Carlsen, Celia Kjaerby, and Nathan Anthony Smith. Astrocytes: integrators of arousal state and sensory context. *Trends in neurosciences*, 46(6):418–425, 2023.

[29] Andrej Karpathy. Nanogpt. https://github.com/karpathy/nanoGPT, 2023.

[30] William Shakespeare. The complete works of william shakespeare. Public Domain Text, 1623. Used as a dataset for next-token prediction in modern machine learning experiments.

[31] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. Jax: composable transformations of python+numpy programs, 2018.

[32] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[33] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[34] Christian Pehle, Sebastian Billaudelle, Benedikt Cramer, Jan Kaiser, Katharina Schreiber, Yannik Stradmann, et al. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16:795876, 2022.

[35] Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning-keynote presentation. In *Communicating Process Architectures 2017 & 2018*, pages 277–280. IOS Press, 2019.

[36] Oliver Richter, Chen Wu, Adrian M Whatley, Georg Köstinger, Claus Nielsen, Ning Qiao, and Giacomo Indiveri. Dynap-se2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor. *Neuromorphic Computing and Engineering*, 4(1):014003, 2024.