COHERENT PREDICTIVE CODING: STABILIZING LATENT-STATE INFERENCE AND LOCAL PARAMETER LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Predictive coding (PC) is a biologically inspired alternative to backpropagation that enables local and parallelizable learning by iteratively inferring latent states and learning parameters from layerwise prediction errors. However, forming useful prediction errors within a limited inference budget remains challenging. We address this problem as two connected stages: latent-state inference, which forms prediction errors, and local parameter learning, which translates them into parameter updates. In Standard PC, latent-state updates depend on learned Jacobians evaluated at the current latent states. After inference, anisotropic activation statistics can distort how local parameter gradients are expressed as block-output changes. We propose **Coherent Predictive Coding (CoPC)**, which fixes feed-forward reference activations a_l and represents latent states as deviations $d_l = h_l - a_l$, yielding $e_l = d_l - d_{l-1}$ so that intermediate inference uses fixed identity coupling rather than learned block Jacobians evaluated at the evolving latent states. After inference, CoPC computes local parameter gradients from the final prediction errors and preconditions each selected linear projection using its uncentered activation second moment to reduce function-space distortion. Across MNIST, Fashion-MNIST, CIFAR-10, and CIFAR-100, CoPC achieves competitive predictive-coding performance and reaches $27.12 \pm 1.01\%$ test accuracy on CIFAR-100, exceeding the strongest predictive-coding baseline in our comparison by 3.25 percentage points. Controlled analyses further show that CoPC produces more consistent latent-state update directions and that second-moment preconditioning improves the alignment between final prediction errors and the induced block-output changes.

1 INTRODUCTION

Predictive coding (PC) originated from the idea that sensory systems remove predictable components of their inputs and transmit prediction errors (Srinivasan et al., 1982), and was later developed into computational accounts of cortico-cortical interactions and hierarchical visual processing (Mumford, 1992; Rao & Ballard, 1999). It was subsequently extended to perceptual inference and learning in hierarchical generative models and related to cortical microcircuits (Friston, 2005; Bastos et al., 2012). More recently, these principles have been implemented as neural network learning algorithms, motivating studies of their relationship to backpropagation (BP) and their empirical learning behavior (Whittington & Bogacz, 2017). From a machine learning perspective, PC is appealing because it implements credit assignment through iterative latent-state inference and local prediction-error learning. Each latent state is updated using prediction errors from adjacent layers, while parameter updates can likewise use errors computed within each layer, allowing local computation and simultaneous state updates within an inference step (Whittington & Bogacz, 2017; Salvatori et al., 2024). Several PC variants can also approximate BP gradients or recover the same gradients under specific conditions (Millidge et al., 2022; Rosenbaum, 2022).

Despite these advantages, training predictive-coding networks remains challenging. Standard PC separates latent-state inference from parameter learning, iteratively updating latent states before computing local parameter updates (Whittington & Bogacz, 2017; Rosenbaum, 2022). The efficiency of PC learning therefore depends on the cost of this iterative inference and on how effectively

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.

the task signal reaches each layer through the latent states (Salvatori et al., 2024; Goemaere et al., 2026). Two central questions then arise: **what inference dynamics emerge when only a practical number of inference steps is available, and how effectively do the resulting prediction errors translate into parameter updates?** Recent work has addressed these challenges from several directions, including changes to the schedule of inference and parameter updates, reformulations of feed-forward predictions or the inference objective, depth-dependent parameterizations, and optimization in prediction-error space (Salvatori et al., 2024; Millidge et al., 2022; Innocenti et al., 2025; Ha et al., 2026; Goemaere et al., 2026).

We view predictive-coding learning as a connected process that forms prediction errors through latent-state inference and translates those errors into local parameter updates. In Standard PC (Whittington & Bogacz, 2017; Rosenbaum, 2022), a learned nonlinear block plays two roles: it forms the feed-forward representation, and its Jacobian at the current latent state determines how prediction errors from adjacent layers contribute to state updates. Because this Jacobian is re-evaluated at every inference step, the update direction changes as the latent states evolve, and the prediction errors formed within a limited inference budget depend on the trajectory taken to reach them. Forming useful prediction errors, however, is not sufficient. After inference, these errors must be converted into parameter gradients for each block, and the activation statistics of the block scale the resulting output change unevenly across directions. This motivates treating the two stages, prediction-error formation during inference and the translation of those errors into parameter updates, as a single connected process.

To this end, we propose **Coherent Predictive Coding (CoPC)**. CoPC first computes feed-forward reference activations a_l and represents each latent state by its deviation $d_l = h_l - a_l$. Its layerwise prediction error then reduces to $e_l = d_l - d_{l-1}$, so intermediate inference uses fixed identity coupling between adjacent latent-state deviations rather than learned block Jacobians evaluated at the evolving states, and the prediction-error term becomes a fixed quadratic form in the deviations. After inference, CoPC computes local parameter gradients from the final prediction errors and preconditions selected linear projections using their uncentered activation second moments. Together, these two stages connect prediction-error formation with local parameter learning.

Contributions.

- We propose a new predictive-coding inference rule that retains learned nonlinear blocks for feed-forward representations and parameter learning while using fixed identity coupling between adjacent latent-state deviations during inference. This construction gives the prediction-error term a fixed quadratic structure that does not involve current-state learned Jacobians.
- We analyze how activation statistics cause local parameter gradients computed from final prediction errors to have different effects across directions, and apply preconditioning based on the uncentered activation second moment of each projection. This preconditioning reduces gain differences across activation eigenmodes. We verify the reduction in directional distortion through an analysis that matches the block-output update norm.
- Through controlled analyses, we isolate the roles of identity coupling and activation second-moment preconditioning, examining latent-state update consistency and prediction-error alignment, respectively. We further evaluate CoPC across four benchmarks and report its predictive performance, per-iteration runtime, and memory cost.

Our experiments examine the role of each design choice. By comparing CoPC with Standard PC and a control that freezes the feed-forward Jacobians, we evaluate whether CoPC reduces changes in latent-state update directions under a limited number of inference steps. A separate analysis measures whether second-moment preconditioning improves alignment between the final prediction error and the block-output change induced by a parameter update. Combining both components, CoPC achieves a test accuracy of $27.12 \pm 1.01\%$ on CIFAR-100, exceeding the strongest PC baseline in our comparison by **3.25 percentage points**. We also evaluate per-training-iteration runtime and peak GPU memory.

2 Background and Related Work

2.1 Inference and parameter learning in predictive coding

For layer l, let h_l denote the latent state and $f_l(\cdot;\theta_l)$ the prediction mapping. We use a convention in which each layer's state is predicted from the preceding layer's state, and define the prediction error as the difference between the latent state and its prediction (Whittington & Bogacz, 2017; Rosenbaum, 2022):

$$\epsilon_l = h_l - f_l(h_{l-1};\theta_l).$$

In a squared-error formulation, these layerwise errors are combined into an objective such as

$$\mathcal{F}_{\mathrm{PC}} = \frac{1}{2}\sum_l \|\epsilon_l\|_2^2$$

and the latent states are iteratively updated by gradient descent on this objective while the parameters remain fixed.

Each latent state contributes to two prediction errors: h_l is both the target predicted from the lower layer and the input used to predict the next layer, h_{l+1}. The negative gradient for an intermediate state therefore generally takes the form

$$-\nabla_{h_l}\mathcal{F}_{\mathrm{PC}} = -\epsilon_l + J_{l+1}(h_l;\theta_{l+1})^\top \epsilon_{l+1},$$

where $J_{l+1} = \partial f_{l+1}/\partial h_l$ is the Jacobian of the prediction mapping evaluated at the current latent state. The first term reduces the current layer's prediction error, while the second transmits the effect of the next layer's prediction error on the current state through the local derivative of the learned mapping. Thus, although the network parameters remain fixed within an inference process, J_{l+1} can change as the latent states evolve.

After inference, prediction errors are recomputed at the final latent states, and parameter gradients are obtained by differentiating each layer's prediction-error loss with respect to that layer's parameters. Parameter updates are computed from local prediction errors with the inferred states held fixed, without differentiating through an unrolled latent-state inference trajectory.

2.2 Improving inference and learning in predictive coding

Recent approaches to improving predictive-coding training have modified different aspects of the learning process, including inference dynamics, learning schedules, optimization variables, and parameterization.

iPC updates parameters during inference, coupling the schedules of latent-state inference and parameter learning (Salvatori et al., 2024).

The Fixed Prediction Assumption (FPA) holds predictions fixed at the feed-forward initialization and constructs inference using local derivatives evaluated at that point (Millidge et al., 2022; Rosenbaum, 2022; 2025). It therefore yields different state dynamics from Standard PC, which reevaluates predictions and Jacobians at the current states at every inference step.

Meta-PCN combines a meta-prediction-error objective with weight-variance normalization to address prediction-error imbalance and nonlinear inference dynamics (Ha et al., 2026). ePC reformulates the computational structure of inference by using prediction errors, rather than latent states, as optimization variables (Goemaere et al., 2026). μPC builds on Depth-μP to parameterize networks as a function of width and depth, addressing the training of residual PC networks and learning-rate transfer across model sizes (Innocenti et al., 2025).

CoPC instead separates the learned representation mapping from the coupling between adjacent latent-state deviations during inference. It retains learned nonlinear blocks for feed-forward representations and parameter learning, while allowing each latent state's deviation from its feed-forward reference activation to contribute directly to the next layer's prediction during inference. Prediction errors can then be expressed as differences between adjacent latent-state deviations, removing the need to repeatedly evaluate current-state learned Jacobians in intermediate latent-state updates.

Section 3 defines this inference rule, the resulting quadratic prediction-error term, and the local parameter learning performed after inference.

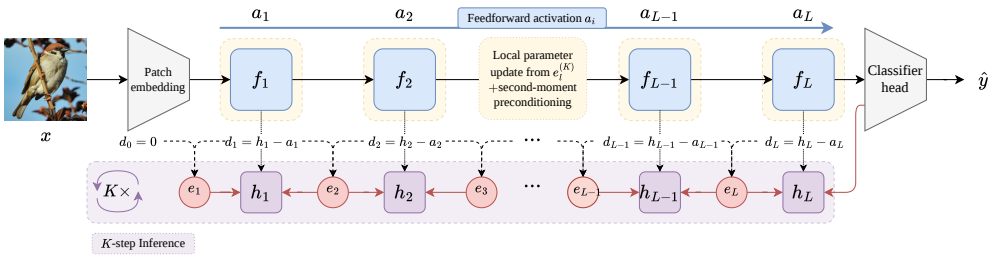

Figure 1: **Overview of Coherent Predictive Coding (CoPC).** A feed-forward pass produces reference activations a_l, and latent states are inferred as deviations $d_l = h_l - a_l$, yielding prediction errors $e_l = d_l - d_{l-1}$ through fixed identity coupling between adjacent latent-state deviations. After inference, the final prediction errors define block-local parameter gradients, and selected linear projections are further adjusted by activation second-moment preconditioning.

2.3 Parameter preconditioning using activation statistics

How a neural network's parameter gradient translates into an output change depends on the activation statistics of the corresponding layer. For example, in a linear projection $Y = XW^\top$, the output change induced by a parameter update is mediated by the input activations X. When activation directions differ substantially in scale, gradient components along those directions can affect the output by different amounts. K-FAC uses activation statistics and output-side gradient statistics to approximate the layerwise structure of the Fisher information (Martens & Grosse, 2015). FOOF implements a gradient step on each layer's neuron activations through a regularized least-squares parameter update, using the inverse input second moment (Benzing, 2022).

In predictive coding, this issue specifically affects the connection between final prediction errors and parameter learning. Even when each block's parameter gradient is computed from its prediction error after inference, anisotropic activation statistics can cause the resulting block-output change to reflect the prediction-error direction unevenly. CoPC uses each linear projection's uncentered activation second moment to regulate this relationship. Section 3 defines the preconditioning rule.

3 Coherent Predictive Coding

Coherent Predictive Coding (CoPC) connects two stages within a single learning rule: latent-state inference that forms prediction errors and local parameter learning that converts the final errors into parameter gradients. The resulting gradients are then preconditioned using the activation statistics of selected linear projections. Throughout this procedure, intermediate latent-state updates depend only on adjacent prediction errors, and block gradients are computed without differentiating through the latent-state inference trajectory. Unless stated otherwise, norms and inner products of state and gradient tensors use the vectorized Frobenius convention.

3.1 Latent-state inference relative to feed-forward reference activations

For an input x, we first perform an ordinary feed-forward pass. For block l with mapping $f_l(\cdot;\theta_l)$, the feed-forward reference activations are given by $a_l = f_l(a_{l-1};\theta_l)$. CoPC holds these reference activations fixed throughout each inference process and initializes the latent states as $h_l^{(0)} = a_l$. Let $d_l = h_l - a_l$ denote the deviation from the reference activation. In CoPC, the prediction and prediction error at layer l are

$$\begin{aligned} p_l &= f_l(a_{l-1};\theta_l) + d_{l-1}, \\ e_l &= h_l - p_l = d_l - d_{l-1}, \end{aligned} \qquad d_0 = 0. \tag{1}$$

The learned nonlinear block determines the feed-forward reference activation, whereas the preceding latent-state deviation enters the next prediction through a fixed identity mapping. We refer to

this interaction as *fixed identity coupling*; consequently, intermediate-layer prediction errors are differences between adjacent latent-state deviations.

For supervised learning, we jointly minimize the prediction-error term and the task loss:

$$\mathcal{F}_{\mathrm{CoPC}} = \mathcal{L}_{\mathrm{task}}(g(h_L;\phi), y) + \frac{\kappa}{2B}\sum_{l=1}^{L}\|e_l\|_F^2, \tag{2}$$

where $\mathcal{L}_{\mathrm{task}}$ is the minibatch-mean cross-entropy, B is the minibatch size, and κ weights the prediction-error term. By Eq. (1), the prediction-error term excluding the task loss is a fixed quadratic form in the latent-state deviations, whose Hessian is independent of the current latent states and learned block Jacobians. The full supervised objective remains nonlinear through the task loss evaluated at the final latent state. Appendix A derives the matrix form and spectrum of the quadratic prediction-error Hessian.

CoPC optimizes Eq. (2) using K synchronous gradient updates. At each inference step, gradients for all layers are computed at the same $h^{(t)}$, after which the states are updated simultaneously. For intermediate layers, the difference between two adjacent prediction errors determines the update; the final latent state additionally receives the task-loss gradient. In compact form, the update is $h_l^{(t+1)} = h_l^{(t)} - \eta_l \nabla_{h_l}\mathcal{F}_{\mathrm{CoPC}}$.

After K inference steps, prediction errors are recomputed at the final states $h^{(K)}$.

3.2 Local parameter learning from final prediction errors

For parameter learning, the final inferred states and reference activations are held fixed, and each block's prediction-error loss is differentiated with respect to that block's parameters. For layer l, let $q_l = \kappa e_l^{(K)}$ denote the weighted prediction error. The raw parameter gradient is

$$G_{\theta_l}^{\mathrm{raw}} = -\frac{1}{B} J_{\theta_l} f_l(a_{l-1};\theta_l)^\top q_l. \tag{3}$$

In Eq. (3), a_{l-1}, $h^{(K)}$, and q_l are treated as constants during parameter differentiation, while the parameter dependence of $f_l(a_{l-1};\theta_l)$ is retained. Each block's gradient is therefore computed within that block's computation graph, without differentiating through the K latent-state inference steps. The task head takes the final latent state $h_L^{(K)}$ as input to compute the ordinary task gradient.

3.3 Activation second-moment preconditioning

The extent to which a parameter gradient derived from a prediction error affects a linear projection's output depends on the statistics of its input activations. CoPC applies activation second-moment preconditioning to selected linear projections. Consider a projection $Y = XW^\top$, where the rows of X collect the activations actually supplied to that projection, and G^{raw} is the raw gradient of its weight matrix obtained from Eq. (3). CoPC uses the uncentered activation second moment

$$C = \frac{1}{n} X^\top X \tag{4}$$

where n is the number of valid-token input rows collected for the projection. CoPC transforms the gradient as follows:

$$\begin{aligned} \bar{G} &= G^{\mathrm{raw}}(C + \lambda I)^{-1}, \\ G^{\mathrm{CoPC}} &= \frac{\|G^{\mathrm{raw}}\|_F}{\max(\|\bar{G}\|_F, \delta_{\mathrm{num}})}\bar{G}. \end{aligned} \tag{5}$$

Here, λ is the ridge parameter and δ_{num} a numerical safeguard; the regularized system is solved without explicitly forming an inverse. For numerically resolved gradients, the rescaling approximately preserves the raw Frobenius norm up to floating-point rounding while retaining directional

Table 1: Official-test accuracy and CIFAR-100 computational cost. Values are mean ± sample SD over three runs under the common protocol in Section 4.1. Runtime and peak memory are measured in a separate CIFAR-100 microbenchmark. Bold and underlined values denote the best and second-best predictive-coding methods; BP is shown as a conventional reference.

Method	Official test accuracy (%) ↑				CIFAR-100 compute ↓	
	MNIST	Fashion-MNIST	CIFAR-10	CIFAR-100	Time / iter. (ms)	Peak memory (GiB)
BP	95.77 ± 0.22	85.23 ± 0.74	64.02 ± 0.50	32.11 ± 0.44	34.47	2.075
Standard PC	19.10 ± 16.02	30.05 ± 12.60	15.05 ± 4.69	2.67 ± 1.94	278.71	3.436
ePC-T	69.24 ± 30.12	80.55 ± 2.18	46.08 ± 7.29	18.77 ± 15.39	<u>177.24</u>	**1.782**
Meta-PCN	<u>71.59 ± 30.67</u>	<u>81.48 ± 1.29</u>	**52.10 ± 1.65**	<u>23.87 ± 0.47</u>	3015.75	3.629
μPC	11.27 ± 1.29	34.82 ± 13.37	26.48 ± 8.59	22.80 ± 0.67	253.04	3.294
CoPC	**92.44 ± 0.17**	**82.57 ± 0.57**	<u>50.42 ± 0.30</u>	**27.12 ± 1.01**	**112.28**	<u>2.082</u>

reweighting. It acts on the gradient passed to the optimizer, not the final optimizer-step norm; all remaining parameters use raw gradients, after which a single optimizer step is taken (Appendix B).

4 Experiments

We evaluate CoPC through a main predictive-coding benchmark and controlled analyses of its latent-state inference and activation second-moment preconditioning, while also reporting computational cost.

4.1 Experimental setup

We evaluate CoPC on MNIST (LeCun et al., 1998), Fashion-MNIST (Xiao et al., 2017), and CIFAR-10 and CIFAR-100 (Krizhevsky, 2009). The main benchmark uses a common ViT-8 architecture family, fixed data splits, 20 training epochs, batch size 256, AdamW with learning rate 2.7×10^{-3}, and three independent seeds. All predictive-coding methods use $K = 8$ inference steps while retaining their method-specific inference rules and parameterizations. Method-specific inference step sizes are selected using training data only and fixed before validation and test evaluation. The outer AdamW setting is shared across methods; accordingly, Table 1 should be interpreted as a controlled common-recipe comparison rather than an estimate of each method's individually optimized performance ceiling.

Our baselines represent distinct approaches to predictive-coding training. Standard PC serves as the reference for state-based iterative inference (Whittington & Bogacz, 2017). ePC-T adapts ePC's error-variable inference to Transformers (Goemaere et al., 2026). Meta-PCN addresses prediction-error imbalance and exploding or vanishing prediction errors through a meta-prediction-error objective and weight-variance normalization (Ha et al., 2026). μPC addresses residual PC network training and learning-rate transfer through width- and depth-dependent parameterization (Innocenti et al., 2025). Together, these baselines cover changes to the inference formulation, prediction-error dynamics, and network parameterization. Backpropagation (BP) is included as a conventional-training reference (Rumelhart et al., 1986).

ePC-T, Meta-PCN, and μPC are adapted to the common ViT supervised setting. Appendix B documents the implementation, baseline adaptations, dataset preprocessing, and hyperparameter selection, while Appendix C provides the evaluation and controlled-analysis protocols. Main-benchmark results are reported as mean ± sample SD over the three independent runs using fixed epoch-20 checkpoints and ordinary feed-forward test predictions. For component analyses, we hold the architecture and data conditions fixed and vary only the inference rule or preconditioning under study.

4.2 Predictive-coding performance and computational cost

Table 1 reports official-test accuracy on all four datasets and per-training-iteration runtime and peak GPU memory on CIFAR-100. BP achieves the highest test accuracy on all four datasets. Among predictive-coding methods, CoPC performs best on MNIST, Fashion-MNIST, and CIFAR-100, and is second to Meta-PCN on CIFAR-10. On CIFAR-100, CoPC reaches $27.12 \pm 1.01\%$, exceeding

Meta-PCN, the strongest PC baseline in this comparison, by **3.25 percentage points**. Appendix Table 3 reports cross-entropy under the same test protocol.

The computational measurements use a separate CIFAR-100 training-iteration microbenchmark under the same hardware and batch setting. CoPC avoids repeated evaluation of nonlinear Transformer blocks during latent-state updates, while parameter learning adds projection-wise activation second-moment solves. Among the evaluated predictive-coding methods, CoPC has the shortest per-iteration runtime at **112.28 ms**, and its peak allocated memory is **2.082 GiB**, second only to ePC-T at 1.782 GiB. Overall, CoPC combines strong predictive-coding performance with relatively low per-iteration cost. We next isolate the contributions of its inference rule and activation second-moment preconditioning.

4.3 Fixed identity coupling stabilizes latent-state updates across inference steps

Standard PC re-evaluates the learned block Jacobian at the current latent state at every inference step. Frozen-J, a mechanism control motivated by the Fixed Prediction Assumption (Millidge et al., 2022; Rosenbaum, 2022; 2025), instead evaluates this Jacobian at the feed-forward reference and keeps it fixed during inference. CoPC removes learned-Jacobian coupling from intermediate latent-state inference: adjacent latent-state deviations interact through fixed identity coupling, while the learned nonlinear blocks remain responsible for feed-forward representation and local parameter learning. Thus, CoPC differs from Frozen-J not merely by freezing the Jacobian, but by replacing adjacent-layer Jacobian-mediated coupling with fixed identity coupling. To isolate the effect of this distinction on training under a limited inference budget, we compare Standard PC, Frozen-J, and CoPC without activation second-moment preconditioning. All three methods use $K = 8$ inference steps and update all latent states simultaneously from the same pre-update states at each step. The inference step size was selected separately for each method using training data (Appendix C.2).

In Figure 2a, CoPC inference achieves a 5-epoch validation accuracy of $9.94 \pm 0.32\%$, substantially exceeding $2.05 \pm 1.37\%$ for Standard PC and $3.50 \pm 1.75\%$ for Frozen-J. We next directly compare the inference dynamics of Frozen-J, which freezes learned Jacobians at the feed-forward reference, and CoPC, which uses fixed identity coupling.

To measure this difference directly, we define each layer's actual latent-state increment as

$$\Delta h_l^{(t)} = h_l^{(t)} - h_l^{(t-1)}$$

and normalize its direction. The *round-to-round update angle* is the angle between the update directions at two consecutive inference steps:

$$\phi_l^{(t)} = \arccos \left\langle s_l^{(t)}, s_l^{(t-1)} \right\rangle, \qquad s_l^{(t)} = \frac{\Delta h_l^{(t)}}{\|\Delta h_l^{(t)}\|_2},$$

All update-direction angles are reported in degrees. A smaller value indicates less rotation of the update direction between steps.

Directional dispersion measures how widely the update directions observed within a layer are distributed around that layer's mean direction. It is computed by averaging the angles between the normalized update directions $s_l^{(t)}$ and the mean direction $\bar{s}_l$. Smaller values indicate that the inference trajectory follows a more consistent direction. The round-to-round update angle is computed only when two consecutive nonzero update directions are both defined, so the first comparable angle in Figure 2b occurs at round 3. Directional dispersion is likewise computed only for intermediate layers with enough numerically resolved update directions; Figure 2c shows only layers that satisfy this condition. Appendix C.2 details the valid-transition criteria and aggregation at the seed level. In Figure 2b–c, CoPC substantially reduces both the round-to-round update angle and directional dispersion relative to Standard PC and Frozen-J. The seed-level round-to-round update angle is $0.279 \pm 0.073^\circ$ for CoPC, compared with $6.34 \pm 6.55^\circ$ for Standard PC and $7.07 \pm 8.79^\circ$ for Frozen-J. Directional dispersion is also lower for CoPC, at $0.575 \pm 0.144^\circ$, compared with $4.240 \pm 1.264^\circ$ for Standard PC and $3.664 \pm 1.985^\circ$ for Frozen-J. These differences persist in intermediate layers with enough observed update transitions.

Thus, fixed identity coupling produces iterative inference dynamics distinct from Jacobian freezing and maintains substantially more consistent latent-state update directions under a limited inference

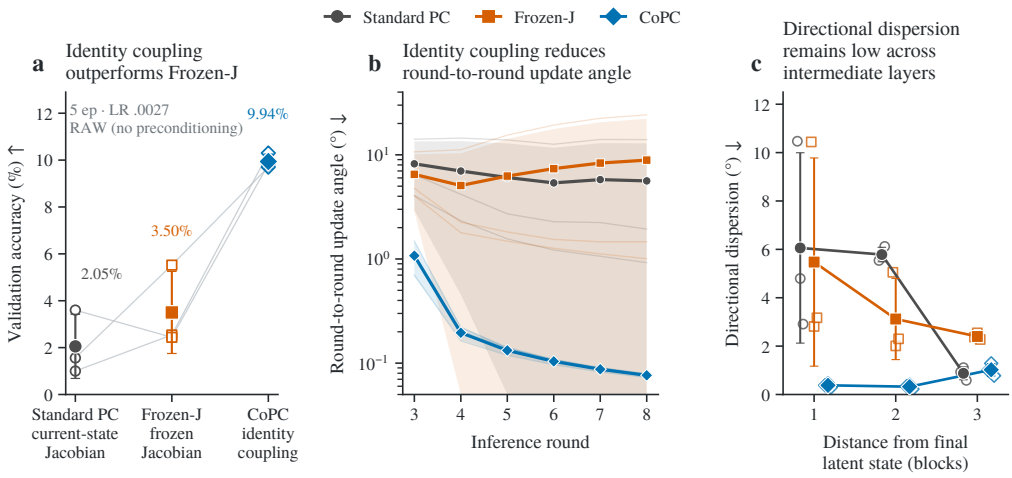

Figure 2: **Identity coupling changes iterative latent-state inference. a,** Five-epoch validation accuracy of Standard PC, Frozen-J, and CoPC without second-moment preconditioning. **b,** Round-to-round update angle over $K = 8$ inference steps, shown from the first step with two resolved consecutive updates. **c,** Directional dispersion across intermediate layers for which sufficient resolved update directions are available, shown by distance from the final latent state. Lower values in **b–c** indicate more consistent update directions. Markers and error bars summarize three independent runs; detailed protocols are given in Appendix C.2.

budget. These differences during inference accompany CoPC's higher performance in controlled training.

4.4 Second-moment preconditioning reduces distortion in local parameter learning

The second stage of CoPC converts the final prediction errors $e_l^{(K)}$ formed during latent-state inference into local parameter updates (Figure 3a). Because anisotropic activation statistics can bias the block-output change induced by a raw parameter update toward particular activation directions, CoPC applies activation second-moment preconditioning to reduce this direction-dependent distortion.

For an activation second-moment eigenvalue ν_i, the function-space mode gain of a RAW update is proportional to ν_i, whereas FULL changes this dependence to

$$\frac{\nu_i}{\nu_i + \lambda},$$

compressing gain differences between modes with large and small second-moment eigenvalues (Figure 3b). To test this effect in nonlinear Transformer blocks, we compare RAW, DIAG, FULL, and ROT at matched block-output update norm; RAW is unpreconditioned, whereas DIAG and FULL use diagonal and full second moments, respectively. ROT preserves FULL's regularized inverse eigenvalue spectrum while changing only its eigenvector orientation (Appendix C.3). This frozen diagnostic holds the prediction-error signal fixed to isolate the directional effect of parameter preconditioning rather than compare complete training procedures.

In this analysis, FULL achieves a seed-level prediction-error alignment of 0.04143 ± 0.00135, compared with 0.01353 ± 0.00035 for RAW, 0.01624 ± 0.00033 for DIAG, and 0.01359 ± 0.00034 for ROT. When paired by seed, block, checkpoint, and minibatch, FULL has higher alignment in 139 of 144 FULL–ROT comparisons (Figure 3c). Because FULL and ROT share the same eigenvalue spectrum, the drop under ROT shows that the actual second-moment orientation contributes to reducing directional distortion in function space.

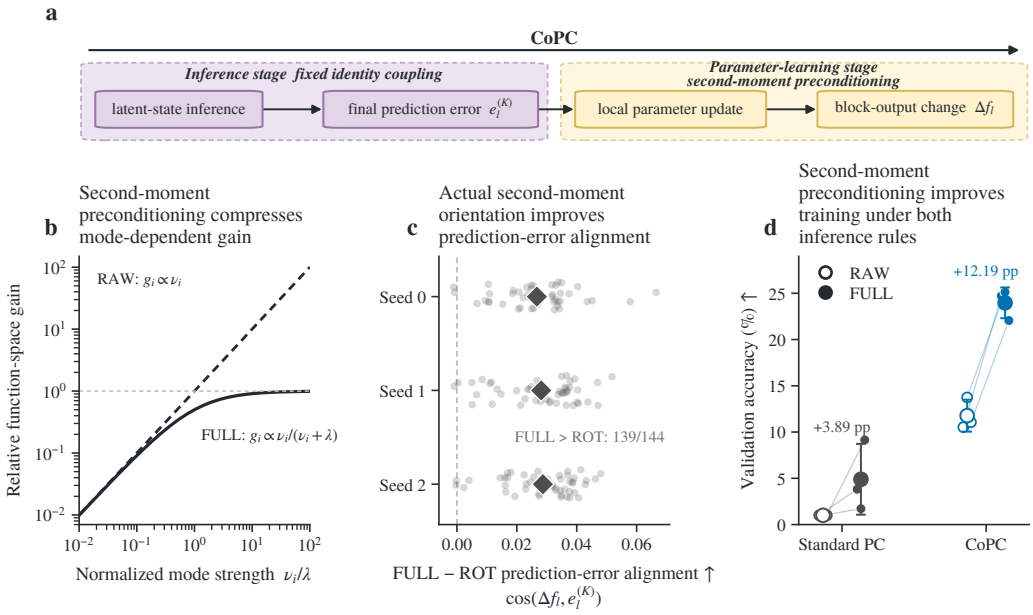

Figure 3: **Two stages of CoPC: latent-state inference and second-moment-preconditioned local learning. a,** CoPC pipeline from latent-state inference to final prediction error $e_l^{(K)}$, local parameter update, and block-output change Δf_l. **b,** Relative function-space gain of RAW and FULL across activation second-moment modes. **c,** FULL–ROT difference in prediction-error alignment at matched block-output update norm; diamonds denote seed-level means. **d,** Five-epoch validation accuracy with RAW and FULL under Standard PC and CoPC inference. See Appendix C.3 for analysis details.

Panel c isolates the directional effect of preconditioning under a matched-update diagnostic, whereas panel d reports validation accuracy in a separate five-epoch component experiment. Adding FULL improves five-epoch validation accuracy by $+3.89$ pp for Standard PC and $+12.19$ pp for CoPC inference (Figure 3d); for CoPC, accuracy increases from $11.77 \pm 1.72\%$ with RAW to $23.97 \pm 1.66\%$ with FULL. A block-range ablation further shows that this training benefit is not confined to preconditioning of the final block (Appendix Table 4). The five-epoch component analyses and the fixed epoch-20 main benchmark use distinct solver parameterizations and independently trained trajectories. Together, these results support CoPC's two-stage design: fixed identity coupling shapes prediction-error formation during latent-state inference, while second-moment preconditioning improves how the resulting local parameter updates are expressed in function space.

5 Conclusion

We introduced Coherent Predictive Coding (CoPC), which jointly designs latent-state inference for prediction-error formation and local parameter learning from those errors. CoPC uses fixed identity coupling between deviations from feed-forward reference activations, followed by activation second-moment preconditioning of local gradients. Controlled analyses show more consistent latent-state update directions than Standard PC and Frozen-J, while preconditioning compresses mode-dependent function-space gains and improves prediction-error alignment. Across four benchmarks, CoPC leads the compared PC methods on MNIST, Fashion-MNIST, and CIFAR-100; on CIFAR-100, it reaches $27.12 \pm 1.01\%$, 3.25 pp above the strongest PC baseline in our comparison. Together, these results suggest that jointly shaping prediction-error formation and local parameter learning is a promising direction for predictive-coding training under limited inference budgets.

AI USE STATEMENT

We used generative AI tools (such as ChatGPT, Codex, and Prism) for editorial assistance, code debugging, literature organization, and translation. All AI-generated or refined content, including code, text, and figures, was thoroughly reviewed, verified, and finalized by the authors, who take full responsibility for the integrity and results of this work.

REPRODUCIBILITY STATEMENT

We provide anonymized source code, experiment configurations, and figure-generation scripts at `https://anonymous.4open.science/r/CoPC/`. The complete CoPC update rule and mathematical derivations are given in Section 3 and Appendix A. Architecture, dataset preprocessing, optimization settings, baseline adaptations, and hyperparameter selection are documented in Appendix B. The protocols used for the main evaluation and the controlled analyses in Figures 2 and 3 are described in Appendix C.

ETHICS STATEMENT

This work studies learning algorithms using publicly available image classification benchmarks and does not involve human subjects or personally identifiable data. We are not aware of ethical concerns specific to the proposed method beyond those generally associated with machine-learning classification systems.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016. doi: 10.48550/arXiv.1607.06450. URL https://arxiv.org/abs/1607.06450.

Andre M. Bastos, W. Martin Usrey, Rick A. Adams, George R. Mangun, Pascal Fries, and Karl J. Friston. Canonical Microcircuits for Predictive Coding. *Neuron*, 76(4):695–711, 2012. doi: 10.1016/j.neuron.2012.10.038. URL https://doi.org/10.1016/j.neuron.2012.10.038.

Frederik Benzing. Gradient Descent on Neurons and its Link to Approximate Second-order Optimization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 1817–1853. PMLR, 2022. URL https://proceedings.mlr.press/v162/benzing22a.html.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *The Ninth International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.11929.

Karl Friston. A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456):815–836, 2005. doi: 10.1098/rstb.2005.1622. URL https://doi.org/10.1098/rstb.2005.1622.

Cédric Goemaere, Gaspard Oliviers, Rafal Bogacz, and Thomas Demeester. ePC: Fast and Deep Predictive Coding in Digital Simulation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2505.20137.

Myoung Hoon Ha, Hyunjun Kim, Yoondo Sung, Youngha Jo, Min S. Kang, and Sang Wan Lee. Stable and Scalable Deep Predictive Coding Networks with Meta-Prediction Errors. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/319e084cd710c3db50a9f9dd7a02a9a8-Abstract-Conference.html.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016. doi: 10.48550/arXiv.1606.08415. URL https://arxiv.org/abs/1606.08415.

Francesco Innocenti, El Mehdi Achour, and Christopher L. Buckley. μPC: Scaling Predictive Coding to 100+ Layer Networks. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/7ea5ad70c984262c064ee9859f7e5f56-Abstract-Conference.html.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *The Seventh International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

James Martens and Roger Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2408–2417. PMLR, 2015. URL https://proceedings.mlr.press/v37/martens15.html.

Beren Millidge, Alexander Tschantz, and Christopher L. Buckley. Predictive Coding Approximates Backprop Along Arbitrary Computation Graphs. *Neural Computation*, 34(6):1329–1368, 2022. doi: 10.1162/neco_a_01497. URL https://doi.org/10.1162/neco_a_01497.

D. Mumford. On the computational architecture of the neocortex. II. The role of cortico-cortical loops. *Biological Cybernetics*, 66(3):241–251, 1992. doi: 10.1007/BF00198477. URL https://link.springer.com/article/10.1007/BF00198477.

Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999. doi: 10.1038/4580. URL https://www.nature.com/articles/nn0199_79.

Robert Rosenbaum. On the relationship between predictive coding and backpropagation. *PLOS ONE*, 17(3):e0266102, 2022. doi: 10.1371/journal.pone.0266102. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0266102.

Robert Rosenbaum. Correction: On the relationship between predictive coding and backpropagation. *PLOS ONE*, 20(3):e0320944, 2025. doi: 10.1371/journal.pone.0320944. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0320944.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

Tommaso Salvatori, Yuhang Song, Yordan Yordanov, Beren Millidge, Lei Sha, Cornelius Emde, Zhenghua Xu, Rafal Bogacz, and Thomas Lukasiewicz. A Stable, Fast, and Fully Automatic Learning Algorithm for Predictive Coding Networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/554414e570a85eb3118e988c5d77986f-Abstract-Conference.html.

M. V. Srinivasan, S. B. Laughlin, and A. Dubs. Predictive coding: a fresh view of inhibition in the retina. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 216(1205):427–459, 1982. doi: 10.1098/rspb.1982.0085. URL https://doi.org/10.1098/rspb.1982.0085.

James C. R. Whittington and Rafal Bogacz. An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural Computation*, 29(5):1229–1262, 2017. doi: 10.1162/NECO_a_00949. URL https://doi.org/10.1162/NECO_a_00949.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017. doi: 10.48550/arXiv.1708.07747. URL https://arxiv.org/abs/1708.07747.

A Mathematical Derivations

A.1 Quadratic prediction-error term and latent-state coupling matrix

As in the main text, let a_l denote the feed-forward reference activation and define the latent-state deviation as

$$d_l = h_l - a_l, \qquad d_0 = 0$$

Stack the layerwise deviations as $d = (d_1^\top, \dots, d_L^\top)^\top$ and define

$$D_L = \begin{bmatrix} 1 & & & 0 \\ -1 & 1 & & \\ & \ddots & \ddots & \\ 0 & & -1 & 1 \end{bmatrix}$$

The CoPC prediction errors can then be written as

$$e = (D_L \otimes I)d$$

Thus, the quadratic prediction-error term, excluding the task loss, is

$$\mathcal{F}_{\mathrm{PE}}(d) = \frac{\kappa}{2B} \left\| (D_L \otimes I)d \right\|_2^2, \tag{6}$$

with Hessian

$$H_{\mathrm{PE}} = \frac{\kappa}{B} \left(D_L^\top D_L \otimes I \right). \tag{7}$$

For a given L and κ, this latent-state coupling matrix is independent of the current latent states, feed-forward reference activations, and learned block parameters.

Let $T_L = D_L^\top D_L$. Then T_L is tridiagonal, with diagonal entries $(2, \dots, 2, 1)$ and adjacent off-diagonal entries -1. Its eigenvalues are

$$\lambda_j(T_L) = 2 - 2\cos\left(\frac{(2j-1)\pi}{2L+1}\right), \qquad j = 1, \dots, L, \tag{8}$$

and

$$\mathrm{cond}_2(T_L) = \frac{\cos^2(\pi/(2L+1))}{\sin^2(\pi/(4L+2))} = \Theta(L^2) \sim \frac{16L^2}{\pi^2}. \tag{9}$$

These properties apply to the quadratic prediction-error term. The full supervised inference objective additionally includes a nonlinear task loss evaluated at the final latent state.

A.2 Local parameter gradients after inference

During parameter differentiation, $h^{(K)}$ and the feed-forward reference activations a are held fixed. The prediction error at layer l can be written as

$$e_l(h; a, \theta_l) = h_l - f_l(\mathrm{sg}(a_{l-1}); \theta_l) - (h_{l-1} - \mathrm{sg}(a_{l-1})), \tag{10}$$

Here, sg denotes stop-gradient. At the current parameters, this value equals $d_l - d_{l-1}$, but its parameter derivative is computed through the learned prediction $f_l(a_{l-1}; \theta_l)$.

With $q_l = \kappa e_l^{(K)}$, the local parameter gradient in the main text is

$$G_{\theta_l}^{\mathrm{raw}} = -\frac{1}{B} J_{\theta_l} f_l(a_{l-1}; \theta_l)^\top q_l. \tag{11}$$

The implementation differentiates $-\langle q_l, f_l(a_{l-1}; \theta_l) \rangle_F / B$ using detached q_l and the feed-forward reference input. Each block's gradient is computed within that block's computation graph, without differentiating through the K latent-state inference steps. The task head computes the ordinary task gradient at detached $h_L^{(K)}$.

A.3 Function-space effects of activation second-moment preconditioning

Consider an affine projection $Y = XW^\top$. For fixed input activations X, let $E = \partial \mathcal{L}_{\text{local}}/\partial Y \in \mathbb{R}^{n \times d_{\text{out}}}$ denote the local loss gradient with respect to the projection output. Then $G^{\text{raw}} = E^\top X$, and a raw gradient step induces the projection output change

$$\Delta Y_{\text{raw}} = -\eta X X^\top E. \tag{12}$$

CoPC uses the uncentered activation second moment

$$C = \frac{1}{n} X^\top X$$

Let ρ denote the Frobenius-norm rescaling factor. The affine output change induced by the preconditioned gradient is

$$\Delta Y_{\text{cond}} = -\eta \rho X (C + \lambda I)^{-1} X^\top E. \tag{13}$$

For the thin SVD $X = U \Sigma V^\top$, let $\nu_i = s_i^2/n$. Then

$$X(C + \lambda I)^{-1} X^\top = nU \operatorname{diag}\left(\frac{\nu_i}{\nu_i + \lambda}\right) U^\top. \tag{14}$$

The function-space mode gain therefore changes from being proportional to ν_i under RAW to being proportional to $\nu_i/(\nu_i + \lambda)$ under FULL. For $\nu_i \geq \nu_j > 0$, the preconditioned gain ratio satisfies

$$\frac{\nu_i}{\nu_j} \frac{\nu_j + \lambda}{\nu_i + \lambda} \leq \frac{\nu_i}{\nu_j},$$

so gain differences across activation modes are reduced. Because ρ is applied uniformly to all modes of a matrix, it does not change this relative gain ratio. We directly measure the output changes of nonlinear blocks in the analysis with matched block-output update norms described in Appendix C.3.

B Implementation and Training Settings

The released code includes the architecture, training configurations, baseline adaptations, and figure-generation scripts described in this section.

B.1 Architecture and datasets

Our backbone, ViT-8, is based on the Vision Transformer (Dosovitskiy et al., 2021) and uses 4×4 patches and eight blocks. Each block consists of a pre-norm attention residual sublayer and a pre-norm MLP residual sublayer with GELU (Hendrycks & Gimpel, 2016). The model has width 128, 4 attention heads, and an MLP expansion ratio of 4. For 32×32 CIFAR inputs, the model uses 64 patch tokens and one class token. MNIST and Fashion-MNIST retain their native 28×28 single-channel inputs without resizing, padding, or RGB replication; a 4×4 patch embedding produces 49 patch tokens plus one class token. The prediction head applies LayerNorm (Ba et al., 2016) and a linear classifier to the final class token. The eight Transformer blocks and final head are trained in all experiments.

The main benchmark uses a training batch size of 256, an evaluation batch size of 512, 20 epochs, and seeds $\{200, 201, 202\}$. The outer optimizer is AdamW (Loshchilov & Hutter, 2019), with a learning rate of 2.7×10^{-3}, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay 10^{-4}, and a constant learning-rate schedule.

Each dataset's official training split is partitioned into stratified 90%/10% training/validation subsets using data seed 0. CIFAR-10 and CIFAR-100 use 45,000/5,000 examples, and Fashion-MNIST and MNIST use 54,000/6,000 examples. CIFAR-10 uses channel-wise normalization with RandomCrop and RandomHorizontalFlip during training. MNIST uses normalization $(0.1307, 0.3081)$ and Fashion-MNIST uses $(0.2860, 0.3530)$, with no data augmentation. CIFAR-100 uses no additional normalization or augmentation beyond $[0, 1]$ scaling. The official test split is not used to select model configurations; it is used only to evaluate epoch-20 checkpoints with an ordinary feed-forward pass.

Local parameter learning and latent-state inference are performed in evaluation mode to disable dropout. Latent states are stored in FP64. Learned Transformer blocks are evaluated in FP32, and their outputs are converted to FP64 for state computations.

B.2 CoPC update sequence

CoPC initializes latent states at the feed-forward reference activations, $h_l^{(0)} = a_l$, and performs $K = 8$ inference steps. All latent-state gradients are computed from the same pre-update states and then applied simultaneously. With $\mathcal{L}_{\text{task}}$ denoting the minibatch-mean task loss, the state gradient used in the implementation is

$$r_l^{(t)} = \begin{cases} \kappa \left(e_l^{(t)} - e_{l+1}^{(t)} \right), & 1 \le l < L, \\ \kappa e_L^{(t)} + B \nabla_{h_L} \mathcal{L}_{\text{task}}(h_L^{(t)}), & l = L, \end{cases} \tag{15}$$

and the state update is

$$h_l^{(t+1)} = h_l^{(t)} - \eta_h r_l^{(t)}. \tag{16}$$

Here, $r_l^{(t)} = B \nabla_{h_l} \mathcal{F}_{\text{CoPC}}$, so the step size in Section 3.1 satisfies $\eta_l = B\eta_h$. The default CoPC configuration uses $\eta_h = 1/32$ and $\kappa = L/(65 \cdot 128) = 1/1040$.

After the final latent-state update, $e^{(K)}$ and $q = \kappa e^{(K)}$ are recomputed at $h^{(K)}$. The gradients for each block and the head are then computed with the feed-forward reference activations and inferred states held fixed. Activation second-moment preconditioning is applied to the selected matrix gradients, followed by a single AdamW step.

B.3 Baseline implementations and ViT adaptations

When the original paper does not use the same ViT architecture, we adapt the method to the common ViT setting and supervised classification while preserving its core learning rule. Each method retains its own inference rule and parameterization in the main benchmark. Using $K = 8$ matches the number of inference steps, not the wall-clock computational cost.

The eight Transformer blocks and task head are included in the optimizer for all methods. The input module is method-dependent: BP updates the patch projection, class token, and positional embedding; Standard PC, CoPC, and Meta-PCN leave these parameters without a local gradient; ePC-T and μPC freeze them explicitly.

Backpropagation (BP). BP updates the input module, all Transformer blocks, and the task head using full-network reverse-mode differentiation of the ordinary mean cross-entropy (Rumelhart et al., 1986). It uses neither separate latent-state inference nor activation second-moment preconditioning.

Standard PC. We apply Standard PC's state-based inference and local parameter learning (Whittington & Bogacz, 2017; Rosenbaum, 2022), treating each Transformer block's output as an independent latent state. At each inference step, nonlinear predictions and input-Jacobian vector products are recomputed at the current states. The latent states are updated simultaneously for $K = 8$ steps. Prediction errors are then recomputed at the final states, and local parameter gradients are calculated using the inferred lower-layer inputs. We do not differentiate through the inference trajectory, and the main benchmark does not use activation second-moment preconditioning. Reference activations are cached under `eval()` and `no_grad`, whereas current-state predictions are recomputed with autograd enabled. In a separate CPU frozen replay, these contexts selected different PyTorch attention execution paths and produced small FP32 initialization residuals; the corresponding residual magnitudes and their effect in the reported GPU training runs were not measured.

Frozen-J. Frozen-J is a mechanism control in Figure 2 and is not included in the main benchmark. It computes

$$J_l^0 = \left. \frac{\partial f_l(h; \theta_l)}{\partial h} \right|_{h = a_{l-1}}$$

once at the feed-forward reference activations, and uses

$$p_l^{\text{FJ}} = a_l + J_l^0 (h_{l-1} - a_{l-1}) \tag{17}$$

during inference. This removes changes in the current-state Jacobian while retaining the learned Jacobian itself. It uses $K = 8$ inference steps with synchronous latent-state updates and no activation second-moment preconditioning. For parameter learning, Frozen-J evaluates block parameter derivatives at the feed-forward lower-state reference a_{l-1}, as in CoPC, while the task head uses the inferred final latent state.

ePC-T. ePC-T is a Transformer adaptation that retains ePC's error-variable inference (Goemaere et al., 2026). It assigns an additive error variable ε_l to each Transformer block and reconstructs the states as

$$s_0 = a_0, \qquad s_l = f_l(s_{l-1}; \theta_l) + \varepsilon_l$$

At each inference step, all ε_l are updated using reverse-mode differentiation through the reconstruction graph. After $K = 8$ steps, block-local outer gradients are computed from the detached reconstructed states. The main benchmark does not use activation second-moment preconditioning.

μPC-ViT. μPC-ViT adapts the width/depth parameterization of the original work (Innocenti et al., 2025) to the common ViT topology. The original paper evaluates residual networks; the attention and classifier scalings are specific to our ViT adaptation. Our implementation applies the specified depth/width scaling to attention logits, residual branches, and classifier logits, and then uses the same current-state inference procedure as Standard PC. It uses the same outer AdamW optimizer as the other methods, without additional per-parameter μP optimizer scaling.

Meta-PCN-ViT. Meta-PCN-ViT adapts the published objective and released implementation to the common ViT setting (Ha et al., 2026). It freezes learned block Jacobians at the feed-forward reference activations. At each inference step, it sequentially updates latent states in two parity groups, recomputing residuals between the two groups. Local block parameters are trained using feed-forward reference inputs and detached inferred states. Meta-PCN's weight-variance normalization is applied after each outer AdamW update. The main benchmark does not use activation second-moment preconditioning.

B.4 Implementation of activation second-moment preconditioning

In each Transformer block, activation second-moment preconditioning is applied to four weight matrices: the combined attention input projection, attention output projection, MLP input projection, and MLP output projection. This gives a total of 32 matrices in ViT-8.

For each matrix, the valid-token activations supplied to the projection are collected in X, with n denoting the number of rows, and

$$C = X^\top X / n$$

is computed in FP32. The base ridge is

$$\lambda_0 = \max\left(10^{-4}\|C\|_F, \epsilon_{\mathrm{FP32}}\right),$$

where ϵ_{FP32} is the FP32 machine epsilon. We solve the regularized linear system by Cholesky factorization without explicitly computing a matrix inverse. If factorization fails, the ridge is increased as $\lambda_j = 10^j \lambda_0$ for $j = 1, 2, 3$.

The Frobenius-norm rescaling factor is computed as

$$\rho = \frac{\|G^{\mathrm{raw}}\|_F}{\max(\|\bar{G}\|_F, \delta_{\mathrm{num}})}, \qquad \delta_{\mathrm{num}} = \mathrm{tiny}_{\mathrm{FP64}},$$

using FP64 norm scalars before casting the scale back to the gradient dtype. For numerically resolved gradients, this approximately preserves the raw Frobenius norm up to floating-point rounding. Biases, LayerNorm parameters, and the task head use raw gradients. This rescaling applies to the matrix gradient passed to AdamW; it does not directly fix the norm of the final optimizer step.

B.5 Hyperparameter selection

The common benchmark learning rate of 2.7×10^{-3} was inherited from a CIFAR-100 train-only calibration using the predefined grid $\{0.0003, 0.0009, 0.0027, 0.0081\}$. The Standard-PC and CoPC

Table 2: Inference step sizes used in the main benchmark.

Method	CIFAR-100	CIFAR-10	Fashion-MNIST	MNIST
Standard PC	$1/64$	$1/32$	$1/32$	$1/64$
ePC-T	0.075	0.075	0.075	0.075
Meta-PCN-ViT	0.8	1.0	0.8	0.8
μPC-ViT	$1/128$	$1/32$	$1/8$	$1/32$
CoPC	$1/32$	$1/64$	$1/32$	$1/64$

Table 3: Test cross-entropy (mean $\pm$ sample SD over independent training runs) under the same official 20-epoch test protocol as Table 1.

Method	MNIST	Fashion-MNIST	CIFAR-10	CIFAR-100
BP	0.1303 ± 0.0053	0.3967 ± 0.0139	0.9894 ± 0.0157	3.2087 ± 0.1379
Standard PC	2.0986 ± 0.3632	1.9558 ± 0.4293	2.6422 ± 0.7331	4.4758 ± 0.1983
ePC-T	0.8358 ± 0.7654	$\underline{0.5412 \pm 0.0629}$	1.4945 ± 0.1970	3.5344 ± 0.9494
Meta-PCN	$\underline{0.7534 \pm 0.7586}$	$\mathbf{0.5138 \pm 0.0406}$	$\mathbf{1.3206 \pm 0.0275}$	$\mathbf{3.0508 \pm 0.0283}$
μPC	2.3447 ± 0.0799	1.7819 ± 0.3306	2.0266 ± 0.1867	$\underline{3.2408 \pm 0.0204}$
CoPC	$\mathbf{0.2559 \pm 0.0152}$	0.6399 ± 0.0304	$\underline{1.3987 \pm 0.0164}$	3.3292 ± 0.1514

source arms selected 2.7×10^{-3} under the training-derived calibration criterion, after which this value was frozen for the common benchmark. It was not jointly optimized across all methods or datasets. The calibration used seed 23001 for 128 updates and the score $(\mathrm{CE}_0 + 2\mathrm{CE}_{64} + \mathrm{CE}_{128})/4$.

Inference step sizes were also selected from predefined, method-specific candidate grids using only training data. Each candidate was run for 2 epochs with a single tuning seed. The primary selection criterion was the normalized trapezoidal area under the training cross-entropy trajectory. Ties were broken first by final training cross-entropy and then by the smaller inference step size. Validation and test results were not used for selection.

For Standard PC and CoPC on CIFAR-100, we retained the settings previously selected using training data, without repeating the search for the final main benchmark. The released code includes the candidate grids and tuning scripts.

C Experimental Protocols

C.1 Main benchmark evaluation and computational cost

Each dataset–method combination is trained with independent seeds $\{200, 201, 202\}$, and we report the mean $\pm$ sample SD. The main benchmark uses ordinary feed-forward predictions from epoch-20 checkpoints, without selecting the best validation checkpoint or using label-dependent inference.

Table 3 reports test cross-entropy under the same protocol as Table 1.

CIFAR-100 runtime and peak GPU memory were measured in a separate training-iteration microbenchmark. We used an NVIDIA GH200 GPU with batch size 256. The framework reported the device string as "NVIDIA GH200 480GB"; the contemporaneous GPU monitor reported approximately 95.6 GiB of GPU memory. For each method, two order-balanced measurement blocks each used 10 warm-up iterations followed by 50 timed iterations, yielding 100 timed iterations in total. Runtime is the median over the pooled measurements with CUDA synchronization around each timing interval and CUDA Event timing. Peak memory was measured with `torch.cuda.max_memory_allocated()` after calling `torch.cuda.reset_peak_memory_stats()` following warm-up. The measurement includes the forward pass, latent-state inference, gradient computation, activation second-moment preconditioning, optimizer step, and ordinary post-step metrics. It excludes model construction, data transfer, validation, checkpointing, and logging. The runtime reported in the main text is therefore a per-training-iteration measurement, not the wall-clock time for full training.

C.2 Inference rules and update-direction metrics

The comparison in Figure 2 evaluates Standard PC, Frozen-J, and CoPC with $K = 8$ inference steps and no activation second-moment preconditioning. All latent-state updates are computed from the same pre-update states and applied simultaneously. The CIFAR-100 inference step size is $1/64$ for Standard PC and $1/32$ for both Frozen-J and CoPC.

Let the actual latent-state increment be

$$\Delta h_l^{(t)} = h_l^{(t)} - h_l^{(t-1)}$$

For numerically nonzero increments, we compute

$$s_l^{(t)} = \frac{\Delta h_l^{(t)}}{\|\Delta h_l^{(t)}\|_2} \tag{18}$$

The round-to-round update angle is defined as the angle between two consecutive update directions:

$$\phi_l^{(t)} = \arccos\left(\mathrm{clip}\left[\left\langle s_l^{(t)}, s_l^{(t-1)}\right\rangle, -1, 1\right]\right)\frac{180}{\pi} \tag{19}$$

For each matched frozen origin, let $\mathcal{T}_l$ denote the inference steps whose layer-l increments are numerically resolved for all compared methods:

$$\mathcal{T}_l = \left\{t : \|\Delta h_{l,m}^{(t)}\|_2 > \epsilon_{64} \max(\|a_l\|_2, 1) \quad \forall m\right\},$$

where ϵ_{64} is the FP64 machine epsilon and m indexes Standard PC, Frozen-J, and CoPC. Directional dispersion is computed only when $|\mathcal{T}_l| \geq 2$ and the resultant mean direction is numerically resolved.

For directional dispersion, the normalized mean of the available update directions in a layer is

$$\bar{s}_l = \frac{\sum_{t\in\mathcal{T}_l} s_l^{(t)}}{\left\|\sum_{t\in\mathcal{T}_l} s_l^{(t)}\right\|_2}$$

and we compute

$$A_l = \frac{1}{|\mathcal{T}_l|}\sum_{t\in\mathcal{T}_l} \arccos\left(\mathrm{clip}\left[\left\langle s_l^{(t)}, \bar{s}_l\right\rangle, -1, 1\right]\right)\frac{180}{\pi} \tag{20}$$

Smaller $\phi_l^{(t)}$ and A_l indicate less rotation between consecutive inference steps and lower dispersion of update directions within a layer, respectively.

The round-to-round update angle requires two consecutive nonzero increments to be defined. Under the common numerical-resolution criterion, comparable angles for block 7 are defined from step 3 onward for all three methods. Earlier values are excluded rather than replaced with $0°$.

For dispersion across depth, we exclude distance 0 because the final latent state has the same update rule in all three methods. With $K = 8$ synchronous latent-state updates, layers farther from the final latent state also lack enough nonzero update directions to compute dispersion. Figure 2c therefore shows only intermediate layers for which the metric is defined; undefined values are not replaced with 0.

Figure 2b–c uses three independent training seeds. For each seed, we rerun the three inference rules on the same training minibatch at seven frozen checkpoints: the common initialization and the epoch-1 and epoch-5 checkpoints of Standard PC, Frozen-J, and CoPC. The round-to-round update angle is computed at the eligible intermediate layer closest to the final latent state. We first average across checkpoints within each seed, then report the mean and sample SD across the three seeds. For directional dispersion, we first average eligible observations within each seed and graph distance, exclude the shared final latent state (distance 0), and then average distances 1–3 with equal weight within each seed. We report the mean and sample SD of these seed-level averages across the three independent seeds.

C.3 Analysis with matched block-output update norms

The mechanism analysis in Figure 3c compares RAW, DIAG, FULL, and ROT using the same prediction-error signal at a fixed checkpoint and minibatch. RAW uses the raw local gradient, while DIAG uses only the diagonal of the activation second moment. FULL uses the full activation second-moment matrix. ROT preserves FULL's regularized inverse eigenvalue spectrum while replacing only the eigenvector orientation with a deterministic orthogonal basis.

In this analysis, we directly compute the actual nonlinear block-output change Δf_l before and after each update. We adjust the scalar multiplier for each condition to match the block-output update magnitude such that

$$\frac{\|\Delta f_l\|_F}{\|e_l^{(K)}\|_F} = 0.01 \tag{21}$$

This matching is an analysis procedure distinct from the pre-AdamW Frobenius-norm rescaling used in training.

Prediction-error alignment is computed as

$$c_l = \frac{\langle \Delta f_l, e_l^{(K)} \rangle_F}{\|\Delta f_l\|_F \|e_l^{(K)}\|_F}, \qquad \text{distortion}_l = 1 - c_l \tag{22}$$

The prediction-error signal in this frozen-checkpoint analysis is generated by a fixed diagnostic procedure that is separate from the main benchmark's $K = 8$ inference procedure with synchronous latent-state updates. After each final-state update, intermediate latent states are updated repeatedly to obtain the analysis signal. These results should therefore not be interpreted as outputs of the same solver used for the main training inference trajectory. The purpose of this distinction is to isolate the directional effect of parameter preconditioning for a shared prediction-error signal, rather than to compare training algorithms.

Paired comparisons cover 3 seeds, 3 checkpoints, 2 training minibatches, and 8 Transformer blocks, yielding 144 FULL–ROT comparisons. Each pair shares the same seed, checkpoint, minibatch, and block, differing only in the conditioning. The "139/144" reported in the main text counts the matched comparisons in which FULL has higher prediction-error alignment than ROT. For statistical aggregation, observations are first averaged within each seed, then the mean and sample SD are computed across the three independent seeds.

The training comparison in Figure 3d evaluates RAW and FULL separately for Standard PC and CoPC inference. All conditions use 5 epochs, $K = 8$ inference steps, a common outer learning rate of 2.7×10^{-3}, and seeds $\{0, 1, 2\}$, and report ordinary validation accuracy. The test split is not used in this component analysis. FULL applies activation second-moment preconditioning to the same 32 projection matrices as CoPC in the main text. RAW uses the same local parameter gradients and AdamW, with only the preconditioning removed.

In the five-epoch component experiments, state updates were written as $h_l \leftarrow h_l - \alpha_l r_l$, where $r_l = B\nabla_{h_l}\mathcal{F}$ and $\kappa = 1/1040$. Standard PC used $\alpha_b = 8.125$ ($\kappa\alpha_b = 1/128$) for non-final latent states and $\alpha_t = 1/256$ for the final latent state. CoPC used $\alpha_b = 32.5$ ($\kappa\alpha_b = 1/32$) for non-final latent states and $\alpha_t = 1/16$ for the final latent state. Both used eight inference steps with synchronous updates of all eight latent states. These coefficients differ from those used in the fixed epoch-20 main benchmark.

Where second-moment preconditioning is applied. To test whether the training benefit is concentrated in the final block, we vary the Transformer blocks to which activation second-moment preconditioning is applied in a five-epoch CIFAR-100 ablation (Table 4). Applying preconditioning to blocks 1–7 yields $24.43 \pm 0.84\%$ validation accuracy, comparable to $23.97 \pm 1.66\%$ when all eight blocks are preconditioned, whereas applying it only to block 8 gives $12.06 \pm 2.03\%$, close to the RAW result of $11.77 \pm 1.72\%$. This indicates that the observed training benefit is not confined to preconditioning of the final block and that non-final blocks contribute materially to the improvement. This ablation changes where activation second-moment preconditioning is applied while keeping all Transformer blocks and the head trainable. It evaluates preconditioning location under the five-epoch controlled configuration and does not establish uniform task-dependent update strength across depth.

Table 4: Five-epoch CIFAR-100 validation accuracy when activation second-moment preconditioning is applied to different Transformer blocks. Values are mean ± SD over three seeds.

Setting	Validation accuracy (%)
RAW	11.77 ± 1.72
Blocks 1–7	$\mathbf{24.43 \pm 0.84}$
Block 8 only	12.06 ± 2.03
All blocks	23.97 ± 1.66