LIGHTWEIGHT AND VERSATILE LEARNED OPTIMIZATION BY RECOMBINATION OF GRADIENT HISTORY

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper presents a lightweight and versatile learned optimizer that dynamically recombines gradient history, represented as averages over disjoint time spans. The optimizer reduces the prediction space to one scalar coefficient per gradient average, shared by multiple parameters. Progressively averaging older gradients minimizes memory cost of long history, while keeping their contributions independently accessible. A 37k-parameter network trained in 0.87 GPU-hours generalizes zero-shot to unseen tasks, lowering validation loss by 9.1% and 0.4% on BERT-Tiny and GPT-Tiny, and improving test accuracy over Adam by 3.5 %p on a Vision Transformer and by 2.7 %p on average across nine graph models, with FLOPs overhead as low as 0.3%.

1 INTRODUCTION

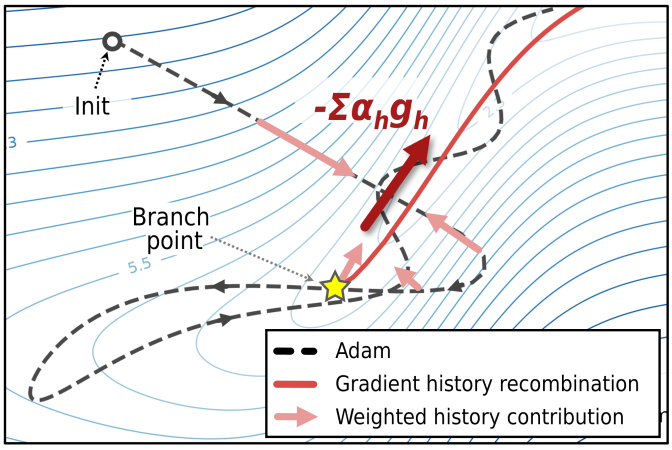

Figure 1: **Recombination of gradient history as a learned update.** Recombination of history contributions $-\sum_h \alpha_h g_h$ with predicted coefficients α_h can redirect the Adam trajectory away from ascent.

While learned optimizers can adapt to loss landscapes, designing lightweight and versatile architectures remains challenging. Coordinate-wise prediction methods (Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2022) are expensive as they process gradient-derived features for individual parameters. For example, training VeLO requires approximately 4,000 TPU-months (Metz et al., 2022), more than twice the compute to train large language models (Chowdhery et al., 2023).

Gradient history can serve directly as a useful directional basis for optimizers, as demonstrated by momentum and Adam (Polyak, 1964; Kingma & Ba, 2014). This motivates an adaptive optimizer that learns to dynamically recombine gradients from the past optimization trajectory.

Building on this principle, this paper presents a learned optimizer that dynamically recombines gradient history, represented by averages over disjoint time spans. To remain lightweight, the optimizer

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.

operates on compact statistics at coarse, configurable parameter granularity. This also allows the optimizer to jointly compare different history components and adjust their relative contributions, while the disjoint representation keeps older time spans independently accessible. Figure 1 visualizes this concept, showing how past gradients are weighted and recombined into each update.

Two techniques make the optimizer versatile and practical. First, fixed-dimensional representation of group statistics supports arbitrary granularity as well as versatility across target networks. Second, progressive averaging of older gradients limits memory growth logarithmically with history length.

The core contributions are:

- The presented learned optimizer justifies the use of gradient-history directions, jointly weighting and dynamically recombining them to construct each update.
- A single trained policy generalizes zero-shot to unseen language, vision, and graph benchmarks, improving validation loss and test accuracy over Adam.
- A compact 37k-parameter policy is trained in under one GPU-hour on an NVIDIA B200.

2 Learned Optimization by Dynamic Recombination of Gradient History

2.1 Dynamic recombination of gradient history

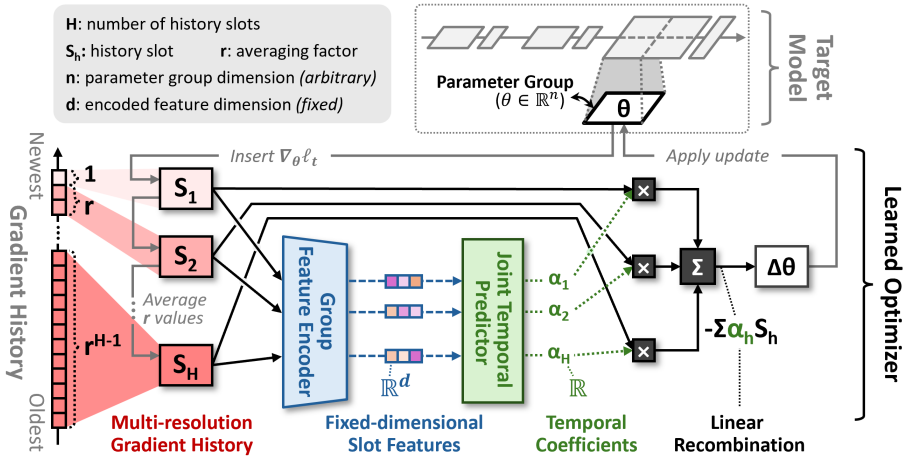

Figure 2: **Dynamic recombination of multi-resolution gradient history.** For each parameter group, the current gradient enters a multi-resolution history. A group feature encoder converts each stored average into a fixed-dimensional slot feature, and a joint temporal predictor assigns the coefficients used to recombine the history into the group update.

The learned optimizer recombines gradient-history averages to construct each parameter update by predicting a coefficient for each average. Target-model parameters are partitioned into disjoint groups, where coordinates within each group share the same temporal coefficient vector.

Figure 2 illustrates this dataflow from gradient-history storage to parameter update. A parameter group b in the target model contains n_b parameters $\theta_t^{(b)}$, and the target-model loss ℓ_t yields the gradient $g_t^{(b)} = \nabla_{\theta_t^{(b)}} \ell_t$ at step t. Each group maintains H history slots for disjoint time spans. $S_{t,h}^{(b)}$ denotes the gradient average stored in slot h, and $\alpha_{t,h}^{(b)}$ its scalar coefficient. Let $\mathcal{A}_t^{(b)}$ denote the set of populated slots. The group update is

$$\Delta\theta_t^{(b)} = -\sum_{h \in \mathcal{A}_t^{(b)}} \alpha_{t,h}^{(b)} S_{t,h}^{(b)}, \qquad \theta_{t+1}^{(b)} = \theta_t^{(b)} + \Delta\theta_t^{(b)}. \tag{1}$$

2.2 MULTI-RESOLUTION GRADIENT HISTORY

Multi-resolution gradient history expands the temporal span covered by a small number of slots. Each history slot is assigned a tier ($c \in \{1, \ldots, C\}$) and can average up to r^{c-1} gradients, where $r \geq 2$ is the scaling factor. As the stored average ages, it is carried forward and averaged into subsequent tiers.

Figure 3 shows an example with $r = 2$ and one slot per tier ($c = h$). At each step, the current group gradient enters slot 1, and its displaced value is carried to slot 2. In subsequent slots, the carried value either fills an empty slot, averages into a partially filled slot, or replaces a full slot while forwarding its previous average. A carry beyond slot H is evicted. Appendix A.1.2 gives the complete update procedure.

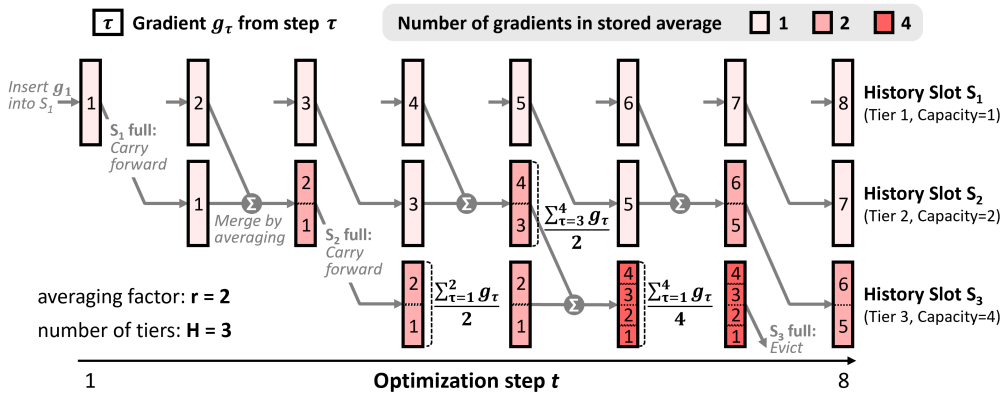

Figure 3: Multi-resolution history with exclusive temporal ranges, tier capacities $1, 2, 4$, and averaging factor $r = 2$. At step 8, carry propagates through full slots S_2 and S_3, evicting the previous average over $g_1, \ldots, g_4$ beyond the final tier.

2.3 TEMPORAL COEFFICIENT PREDICTION

The temporal coefficients are jointly predicted from its current optimization state and stored gradient history. Figure 4 depicts the architecture.

Group feature encoder For each parameter group, the group feature encoder summarizes each history slot and the current parameters into a fixed-dimensional slot feature, using their histograms and other statistics such as group dimension, scale, concentration, and alignment. Separate MLPs project both types of information into $\mathbb{R}^d$, where d is independent of n_b, and a learned two-way Softmax gate fuses them into one slot feature. Appendix A.1.1 provides its architecture and numerical safeguards.

Joint temporal predictor The joint temporal predictor generates the coefficients by processing all slot features via self-attention. Positional encoding provides tier context, including the center and length of each slot's history span. Following self-attention, a coefficient head and Softmax yield nonnegative allocations $a_{t,h}^{(b)}$ that sum to one over $\mathcal{A}_t^{(b)}$. A separate scale token predicts a group-level gain $s_t^{(b)}$, which scales these allocations into the final temporal coefficients $\alpha_{t,h}^{(b)}$ used in Equation 1. Appendix A specifies the evaluated attention configuration.

2.4 EXTENSION ACROSS PARAMETER PARTITIONS

The optimizer architecture can extend to support overlapping parameter groups via an additional spatial recombination process. After temporal recombination, a separate network sharing identical architecture processes the intermediate output of each partition. It then applies spatial coefficients to fuse these outputs into the final parameter step. While the evaluation focuses on a single partition, Appendix C defines and evaluates the extension.

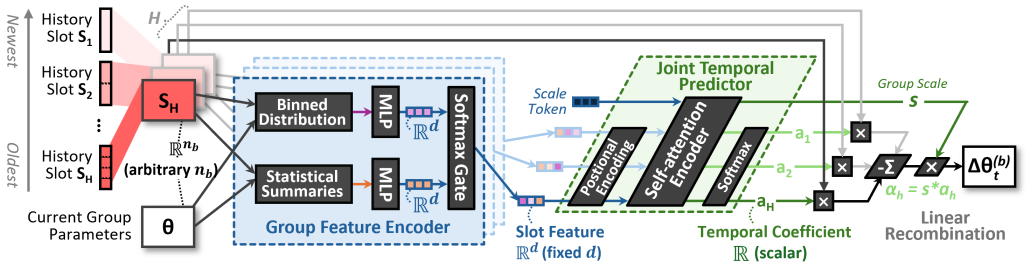

Figure 4: **Temporal coefficient prediction.** For each history slot, the stored gradient average and current group parameters are converted into histogram features and scalar summaries, then fused into a fixed-dimensional slot feature. Positioning encoding identifies the represented range, and self-attention processes the slot features jointly. Softmax produces normalized temporal allocations $a_{t,h}^{(b)}$, while a group scale converts them into the final coefficients $\alpha_{t,h}^{(b)}$ used in Equation 1.

2.5 Trajectory-level optimizer training

The optimizer is trained to minimize the cumulative change in held-out loss over an unrolled trajectory of K updates, rewarding persistent improvements. Loss scales are normalized for each meta-task to contribute equally. Let L_t^{held} be the held-out loss before the unroll and L_{t+k}^{held} the loss after update k. The meta-loss is:

$$\mathcal{L}_{\text{meta}} = \sum_{k=1}^{K} \frac{L_{t+k}^{\text{held}} - L_t^{\text{held}}}{L_t^{\text{held}}}. \tag{2}$$

Appendix A.2.1 specifies the numerical safeguards and exact implementation.

3 Evaluation

3.1 Experimental Setup

Optimizer training. The 37,389-parameter learned optimizer is meta-trained on four small networks (under 10,000 parameters). Six history slots (63-step span) were used during optimizer training, where each meta-epoch consists 160 model-training steps.

Target tasks and zero-shot protocol. The trained policy is evaluated zero-shot on BERT, GPT, Vision Transformer, and GNN architectures unseen during meta-training. The optimizer uses history slots to cover the entire target trajectory, referred to as *full-span* throughout unless otherwise stated.

Baselines and pairing. Adam, stochastic gradient descent (SGD), and SGD with momentum (SGDM) use default hyperparameters to all tasks. This is intended to evaluate off-the-shelf deployment performance of the learned optimizer. Paired comparisons share the same initialization, data ordering, stochastic streams, training budget, and parameter grouping. Target network tensors are partitioned into groups of at most 256 coordinates, prioritizing to share output channel dimension.

Appendix A specifies the group construction, optimizer-training configuration, target protocols, baseline hyperparameters, and metrics.

3.2 Zero-Shot Generalization to Unseen Tasks

Language models. On language models, the learned optimizer successfully generalizes to held-out language tasks 100,000-step horizon despite only being meta-trained seeing 160-steps with six history slots. It reaches the lowest terminal validation loss among all baselines. (Table 1, Figure 5). This section reports the setting which embedding and output-head tensors are delegated to Adam.

Table 1: Overall comparison on tasks unseen during optimizer training. Language models report terminal validation loss. Vision reports test accuracy both at the state with the lowest validation loss and at the terminal state. Graph models report test accuracy from the state with the lowest validation loss within 100 updates on Cora and CiteSeer and 500 updates on PubMed. Bold and underlined values denote the best and second-best result per row. Graph results are mean ± standard deviation over three seeds.

Data	Model	Learned	Adam	SGD	SGDM
Language modeling	*Terminal validation loss* ↓				
Wikipedia–BookCorpus	BERT-Tiny (masked-token)	**3.463***	3.810	6.703	6.222
WikiText-103	GPT-Tiny (next-token)	**4.250***	4.268	5.921	4.686
Vision	*Test accuracy (%)* ↑				
CIFAR-100	ViT (lowest val. loss)	31.13	27.61	17.29	**32.54**
	ViT (terminal)	**31.74**	27.95	27.25	31.22
Graph node classification	*Validation-selected test accuracy (%)* ↑				
Cora	GCN	**81.7 ± .2**	79.3 ± .8	31.8 ± 10.0	74.9 ± .9
	GraphSAGE	**79.6 ± .4**	78.7 ± .3	24.1 ± 3.6	76.6 ± .2
	GAT	**82.0 ± .6**	78.3 ± 1.3	47.2 ± 8.0	79.0 ± 1.2
CiteSeer	GCN	**71.0 ± .8**	66.2 ± 1.6	33.0 ± 6.2	70.6 ± .7
	GraphSAGE	69.9 ± .9	65.6 ± 1.4	34.7 ± 9.4	**70.2 ± .4**
	GAT	69.1 ± .3	66.2 ± .8	52.9 ± 1.4	**70.4 ± .8**
PubMed	GCN	**79.1 ± .2**	77.5 ± .4	51.8 ± 1.2	67.7 ± 6.1
	GraphSAGE	**77.2 ± .1**	75.2 ± .4	50.2 ± 13.7	74.7 ± .4
	GAT	**78.8 ± .2**	77.1 ± 1.4	59.2 ± 6.0	73.3 ± .2
GNN arithmetic mean across 9 cells		**76.47 ± .24**	73.78 ± .26	42.76 ± 2.36	73.04 ± .75

* Embedding and output-head tensors are trained with Adam.

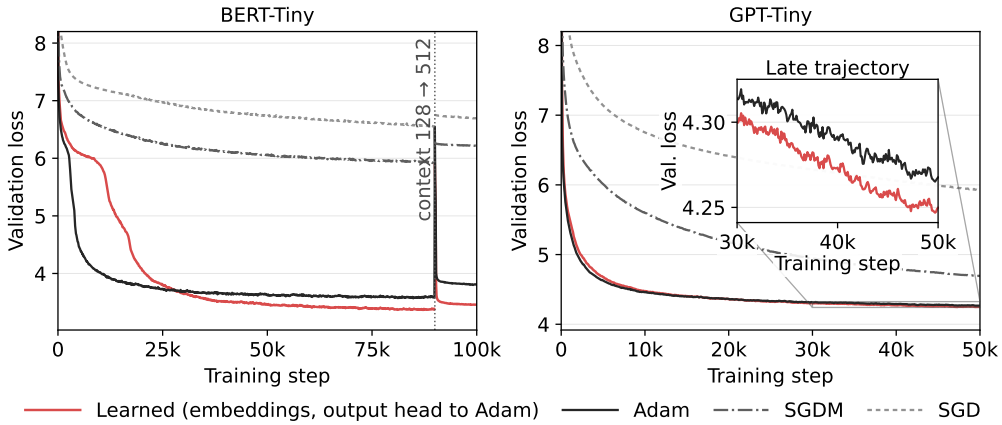

Figure 5: The learned optimizer updates all tensors except the embedding and output-head tensors. Adam trains those tensors using the same hyperparameters as the all-Adam baseline. BERT-Tiny follows a 100,000-update protocol with a context transition at update 90,000, while GPT-Tiny is evaluated at 50,000 updates. The GPT magnifies the later trajectory.

Applying the learned optimizer to all parameters or exclusively to the embedding and output head also maintain convergence and can even outperform the baselines (Appendix B.1.1).

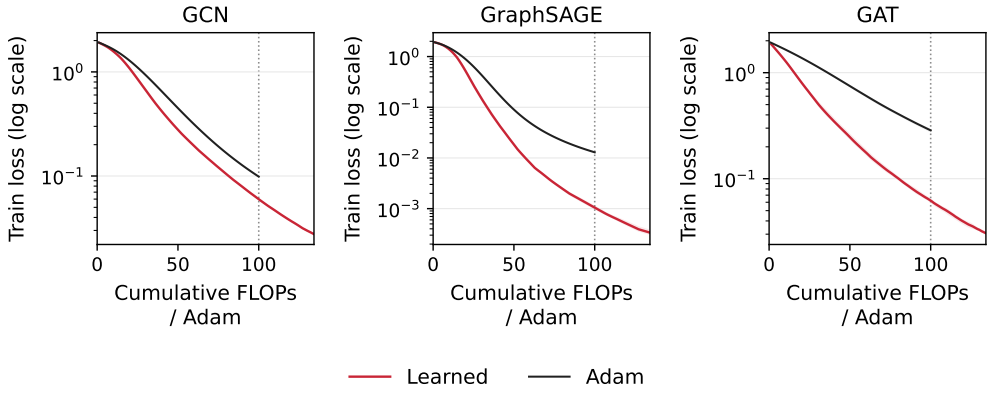

Figure 6: FLOP-matched Cora training loss under full span. The horizontal axis expresses cumulative profiler-visible FLOPs in units of one Adam step, with a budget equal to 100 Adam updates. Lines show three-seed means, and bands show seedwise ranges.

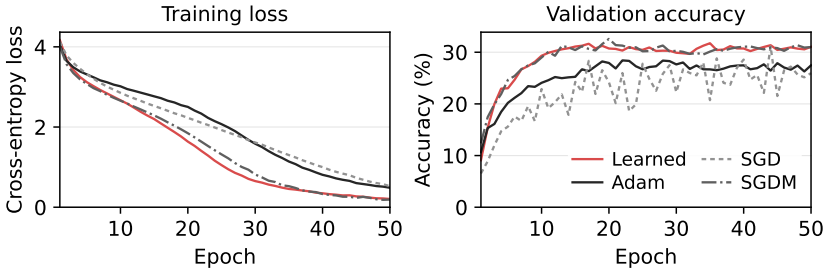

Figure 7: ViT/CIFAR-100 epoch-mean training loss and validation accuracy over 50 epochs under matched initialization, split, and minibatch order. Table 1 reports validation-selected and terminal test accuracy.

Graph models. The learned optimizer further generalizes to unseen graph modalities and node-classication tasks. The learned optimizer is evaluated on Cora, CiteSeer, and PubMed using GCN, GraphSAGE, and GAT Kipf & Welling (2016); Hamilton et al. (2017); Veličković et al. (2018); Sen et al. (2008). It achieves the highest validation-selected test accuracy in seven of nine comparisons and the highest nine-cell arithmetic mean (Table 1). Accounting for inference overhead, it yields lower training loss than Adam at equal cumulative FLOPs on Cora (Figure 6). Appendix B.1.2 reports all FLOP-matched trajectories.

Vision transformer. Training a six-block ViT on CIFAR-100 tests generalization to a larger model scale with unseen image dataset. The learned optimizer outperforms Adam in both validation-selected and terminal test accuracy, achieving the highest accuracy overall (Table 1). Figure 7 reports the training loss and validation accuracy trajectories.

3.3 Ablations and Behavioral Analysis

Temporal range. Restricting history to a 63-step span demonstrates that full-span coverage is not required to maintain convergence across all tasks (Table 2). Full span improves terminal metrics for language and vision models, but yields effectively identical mean accuracy across the nine graph tasks (Appendix B.2.2). These results show that learned optimizer is effective across different history spans, even when the full span is unavailable. Appendix B.2.2 reports the complete GNN comparison.

Temporal dynamics. Figure 8 shows that the learned policy is dynamic and behaves differently from Adam. Following a shared initial trajectory, transitioning to Adam branches to a different solution with higher terminal training loss (Figure 8a). At this branch, weights assigned to the

Table 2: Effect of history span. Language models report terminal validation loss. ViT reports validation-selected and terminal test accuracy, and GNNs report validation-selected test accuracy. The GNN row is the unweighted arithmetic mean across nine model–dataset comparisons, reported as mean ± sample standard deviation across three target seeds. Bold and underlined values mark the best and second-best result in each row.

Target	Metric	Full span	63-step span	Adam
BERT-Tiny	Val. loss ↓	**3.463***	3.824*	<u>3.810</u>
GPT-Tiny	Val. loss ↓	**4.250***	<u>4.261</u>*	4.268
ViT/CIFAR-100	Test acc. (best val.) (%) ↑	**31.13**	<u>30.81</u>	27.61
	Test acc. (terminal) (%) ↑	**31.74**	<u>30.86</u>	27.95
GNN arithmetic mean across 9 cells	Test acc. (%) ↑	<u>76.44 ± 0.28</u>	**76.47 ± 0.24**	73.78 ± 0.26

* Embedding and output-head tensors are trained with Adam.

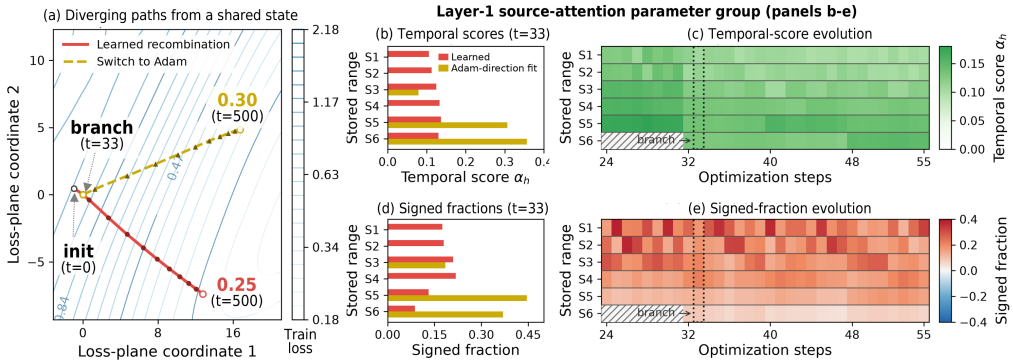

Figure 8: Temporal dynamics on GAT/PubMed. (a) Both paths share the trajectory through $t = 32$, with Adam states reconstructed from the learned path. At step $t = 33$, Adam branches and reaches terminal training loss of 0.30 ($t = 500$), and 0.25 for learned recombination. (b,d) Red bars show the scale-normalized temporal scores and signed contribution fractions for the `layer-1 source-attention` group at branch ($t = 33$). Signed fractions indicate each slot's directional contribution to the constructed update. S_1 corresponds to the slot with newest gradient, and S_6 with the oldest. Gold bars show the closest nonnegative coefficient allocation fitted to the norm-matched Adam direction. (c,e) The learned scores and signed contributions vary over updates 24–55. Dotted boxes mark $t = 33$, and hatching marks unpopulated slots.

history slots by the learned optimizer differs from norm-matched Adam update (Figure 8b,d). The predicted temporal scores and signed contributions vary over the optimization steps, confirming the policy is not static (Figure 8c,e). Appendix B.2.3 shows the behavioral analysis of other parameter groups.

3.4 Computational Cost and Deployment Scaling

Optimizer training and deployment costs. Training the learned policy requires 64 meta-updates and consumes 0.87 NVIDIA B200 GPU-hours. During deployment, routing all parameters through the learned optimizer requires 1.035× and 1.166× the Adam FLOPs per step on BERT and GPT, respectively. The corresponding end-to-end step times are 1.135× and 2.336× the matched Adam baselines (Figure 9).

History slot scaling. In the language profiles, persistent optimizer state covers up to 131,071 past gradients, occupying 301 MB for BERT and 466 MB for GPT. Adam occupies 35 MB and 55 MB in the matched profiles, respectively. Appendix A.4 specifies the profiling boundaries and state accounting.

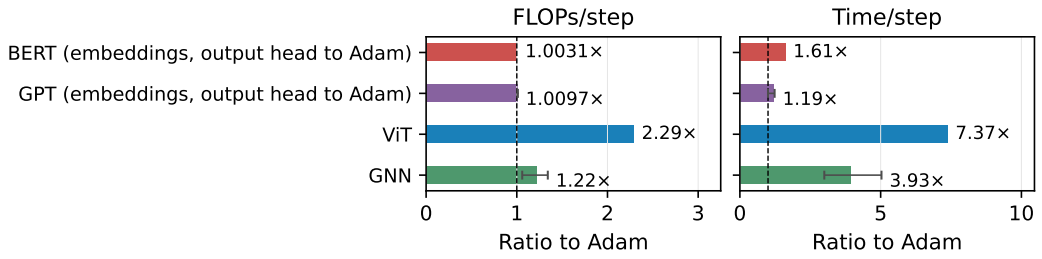

Figure 9: Deployment cost relative to Adam in isolated NVIDIA B200 profiles using bfloat16 arithmetic. A value of 1.0 denotes the matched Adam baseline. FLOPs/step include forward, backward, the learned update, and any update routed to Adam. Time/step measures the same end-to-end boundary. BERT values weight its 128- and 512-token contexts by $0.9/0.1$. GNN values are geometric means over the nine model–dataset tasks, with min–max error bars. Appendix A.4 reports all profiles for the learned optimizer.

4 Conclusion

This paper demonstrates that a compact temporal recombination policy, meta-trained only on small classification tasks, can generalize to unseen tasks. These results establish recombination of gradient history as a practical design principle for learned optimization.

AI use statement

Generative AI tools were used to provide feedback on research methodology and experimental design, assist with software implementation and verification, search and summarize literature, refine scientific figures, and draft and edit manuscript text. They were not used to generate datasets, formulate or prove mathematical claims, translate the manuscript, prepare datasets, or conduct qualitative or thematic data analysis. The authors reviewed the cited sources, experimental outputs, code changes, figures, and manuscript content, and take responsibility for the final text, claims, and artifacts.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113, 2023.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. doi: 10.1016/0041-5553(64)90137-5.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, September 2008. ISSN 0738-4602. doi: 10.1609/aimag.v29i3.2157. URL https://doi.org/10.1609/aimag.v29i3.2157.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 3751–3760. PMLR, 2017.

A Implementation and Evaluation Details

This appendix specifies the evaluated optimizer, optimizer-training procedure, target protocols, and resource measurements.

A.1 Learned-optimizer configuration

Each parameter tensor is partitioned into fixed groups of at most 256 active coordinates. For tensors with at least two dimensions, trailing dimensions are flattened within each leading-dimension slice. Adjacent complete slices are packed up to the limit, and oversized slices are split into contiguous segments. One-dimensional tensors are segmented directly. Final incomplete segments are padded for batched policy inference, with padded coordinates masked from features and updates.

The evaluated optimizer uses six history slots with capacities $1, 2, 4, 8, 16, 32$ and $r = 2$. Manuscript indices run newest to oldest. Attention receives the reverse order. Each populated slot is paired with its temporal center and stored-gradient count.

A.1.1 History-slot feature definitions

For a group of n_b parameters, let $S = S_{t,h}^{(b)}$ denote a populated stored average and $\theta = \theta_t^{(b)}$ its current parameter vector. Table 3 defines the six statistics computed per slot. The stabilized norms are $\mathrm{RMS}_\epsilon(v) = \sqrt{\|v\|_2^2/n_b + \epsilon}$ and $\|v\|_{2,\epsilon} = \sqrt{\|v\|_2^2 + \epsilon}$.

Table 3: Statistics computed for the stored gradient average in each populated history slot.

Feature	Definition
Group size	$\log_{10} n_b$
Stored-average scale	$\log_{10} \mathrm{RMS}_\epsilon(S)$
Current-parameter scale	$\log_{10} \mathrm{RMS}_\epsilon(\theta)$
Coordinate uniformity	$\|S\|_1/(\sqrt{n_b}\|S\|_{2,\epsilon} + \epsilon)$
Gradient–parameter alignment	$\langle S, \theta\rangle/(\|S\|_{2,\epsilon}\|\theta\|_{2,\epsilon} + \epsilon)$
Gradient-to-parameter scale	$\log_{10} \mathrm{RMS}_\epsilon(S) - \log_{10} \mathrm{RMS}_\epsilon(\theta)$

The histogram path normalizes each populated slot by its gradient RMS. For normalized coordinate $\widetilde{S}_i$ and current parameter θ_i, the seven tuple entries are $\mathrm{sign}(\widetilde{S}_i)\log_{10}(1 + |\widetilde{S}_i|)$, $\log_{10}(|\widetilde{S}_i| + \epsilon)$, $\log_{10}(|\theta_i| + \epsilon)$, $\log_{10}((|\widetilde{S}_i| + \epsilon)/(|\theta_i| + \epsilon))$, a signed logarithm of $\widetilde{S}_i$ relative to the slot RMS, $\mathrm{sign}(\widetilde{S}_i)\,\mathrm{sign}(\theta_i)$, and the normalized group index in the extraction batch. Tuple channels are standardized within each slot and clipped to $[-5, 5]$. Sixteen bins are indexed by $\log_{10}(|\widetilde{S}_i| + \epsilon)$ and store their assigned mean tuples. Empty bins are zero. Padded coordinates are excluded, non-finite summaries are replaced by finite sentinels, and feature extraction is detached during optimizer training.

Two-layer MLPs with intermediate LayerNorm and GELU project the six statistics and 112-dimensional histogram to separate 32-dimensional vectors. LayerNorm precedes their learned two-way gated fusion. A fixed sinusoidal encoding adds temporal center and stored-gradient count. Unpopulated slots are excluded from attention and coefficient normalization.

The temporal predictor uses two width-32 Transformer encoder layers, four attention heads, and feed-forward width 128. Softmax normalizes coefficients over populated slots. The evaluated realization also applies an auxiliary group scale after Equation 1. Its learned scale vector attends to every populated slot, while an asymmetric mask prevents slot tokens from attending to it. For its contextualized representation $h_{t,\mathrm{scale}}^{(b)} \in \mathbb{R}^d$ and scalar projection $q : \mathbb{R}^d \to \mathbb{R}$, the scale is

$$s_t^{(b)} = 2\,\sigma\Big(z_0 + \epsilon \tanh\Big(q(h_{t,\mathrm{scale}}^{(b)})\Big)\Big) \in (0, 2),$$

where z_0 is learned and $\epsilon = 0.5$. The scale is initialized at one and multiplies the completed recombination.

Algorithm 1 One-slot-per-tier multi-resolution history update at optimization step t. Processing stops when no value is carried to the next slot.

Require: g_t current parameter-group gradient
$S_{t-1,1:H}$ history slots from step $t-1$
$c_{t-1,2:H}$ slot accumulation counters from step $t-1$
H, r history depth and accumulation factor
Ensure: $S_{t,1:H}, c_{t,2:H}$
$(S_{t,1:H}, c_{t,2:H}) \leftarrow (S_{t-1,1:H}, c_{t-1,2:H})$
carry $\leftarrow S_{t-1,1}$; $S_{t,1} \leftarrow g_t$ {insert the current gradient}
$h \leftarrow 2$
while $h \leq H$ and carry $\neq$ empty **do**
  if $c_{t-1,h} = 0$ **then**
    next $\leftarrow$ empty; $S_{t,h} \leftarrow$ carry; $c_{t,h} \leftarrow 1$ {fill an empty slot}
  else if $c_{t-1,h} = r$ **then**
    next $\leftarrow S_{t-1,h}$; $S_{t,h} \leftarrow$ carry; $c_{t,h} \leftarrow 1$ {replace the full slot and carry its value}
  else
    next $\leftarrow$ empty
    $S_{t,h} \leftarrow \big(c_{t-1,h} S_{t-1,h} + \text{carry}\big)/(c_{t-1,h} + 1)$
    $c_{t,h} \leftarrow c_{t-1,h} + 1$ {merge by averaging}
  end if
  carry $\leftarrow$ next; $h \leftarrow h + 1$
end while
return $S_{t,1:H}, c_{t,2:H}$

A.1.2 Multi-resolution history update

Algorithm 1 gives the update for one group, omitting group indices. $c_{t,h}$ counts values accumulated at tier h. Slots and counters start empty or zero, and a carry beyond tier H is discarded.

A.2 Optimizer-training tasks and protocol

Table 4: Optimizer-training tasks. Parameter counts include all trainable parameters.

Model	Dataset	Architecture	Params.
MLP	Synthetic binary	64 inputs; two 64-unit ReLU layers; 2-class head	8,450
ConvNet	CIFAR-10	5×5 convolutions with 2, 4 channels; ReLU and 2×2 pooling; 16-unit ReLU; 10-class head	2,142
MicroResNet	Fashion-MNIST	8-channel 3×3 BN stem; residual blocks of width 8, 16; adaptive pooling; 10-class head	5,122
MicroViT	CIFAR-10	8×8 patches; width 16; 2 heads; one pre-normalized block; FF width 64; 10-class head	6,810

CIFAR-10 and Fashion-MNIST inputs are normalized channelwise with mean and standard deviation 0.5, without augmentation. MicroViT additionally uses a learned class token, learned positional embeddings, and zero dropout. The synthetic dataset contains 20,000 standardized 64-dimensional examples with 32 informative and 10 redundant features and class separation 0.8. Each training set is split stratified 80/20 into model-training and held-out meta-objective partitions.

A.2.1 Normalized cumulative held-out-loss objective

The complete 20% held-out partition is evaluated after every simulated update in microbatches of 512. The starting loss is detached and the denominator in Equation 2 is lower-bounded by 0.1. Feature statistics are also detached. Gradients pass through every other optimizer path, simulated update, and subsequent held-out loss. Table 5 summarizes the remaining optimizer-training configuration.

Table 5: Optimizer-training configuration.

Item	Value
Tasks per meta-window	All four tasks in Table 4
Model-training batch	128
Reset trajectories / steps per trajectory	4 / 160
Successful meta-update cap	64
TBPTT horizon / meta-update cadence	20 / 10 model-training updates
Meta-objective	Normalized cumulative held-out-loss change in Equation 2
Meta-gradient	full differentiation through exposed paths
Meta-optimizer	Adam, learning rate 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, global gradient-norm clipping at 10
Held-out evaluation microbatch	512
Objective schedule	Model-training task loss only

A.3 Target-model training and evaluation

A.3.1 Shared baseline hyperparameters

Across all evaluation domains, the baseline optimizers use fixed hyperparameters and zero weight decay. Adam uses a learning rate of 10^{-3}, $(\beta_1, \beta_2) = (0.9, 0.999)$, and $\epsilon = 10^{-8}$. SGD and SGDM use a learning rate of 0.01, with momentum 0 and 0.9, respectively. No domain-specific hyperparameter sweeps are applied, as this evaluation specifically tests deployment performance where exhaustive hyperparameter tuning is computationally prohibitive.

A.3.2 Language models

Both language targets use two layers, width 128, and two heads. BERT-Tiny follows Turc et al. (2019). The causal target uses a GPT-2 decoder (Radford et al., 2019). Table 6 summarizes their training configurations.

Table 6: Language-model configurations used for optimizer evaluation.

Model	Layers / width / heads	Context	Global / micro	Data and objective
BERT-Tiny	2/128/2	128 for 90k, then 512 for 10k	256/8	Wikipedia–BookCorpus-compatible masked-token prediction
GPT-style	2/128/2	128 fixed	64/16	WikiText-103 next-token prediction

Embeddings and the output head use Adam. Learned recombination updates all other language-model parameters. Paired runs share initialization, data, stochastic streams, and a 20-microbatch validation panel. Validation restores model mode and random-number state, and loss is aggregated over predicted targets.

A.3.3 Vision-transformer protocol

The CIFAR-100 model uses six Transformer blocks, width 256, eight heads, 8×8 patches, and dropout 0.1. The learned optimizer updates all parameters. A fixed stratified split supplies 45,000 training and 5,000 validation examples, with batch size 64 and 704 updates per epoch. Validation is measured after each of 50 epochs, and the terminal state is evaluated once on the test set.

A.3.4 Graph models

The graph evaluation crosses GCN, GraphSAGE, and GAT with Cora, CiteSeer, and PubMed. Cora and CiteSeer run for 100 updates with seven slots. PubMed runs for 500 updates with nine slots.

The public Planetoid split and stored node features are used without input-feature normalization. Each split has 20 labeled training nodes per class, 500 validation nodes, and 1,000 test nodes, giving

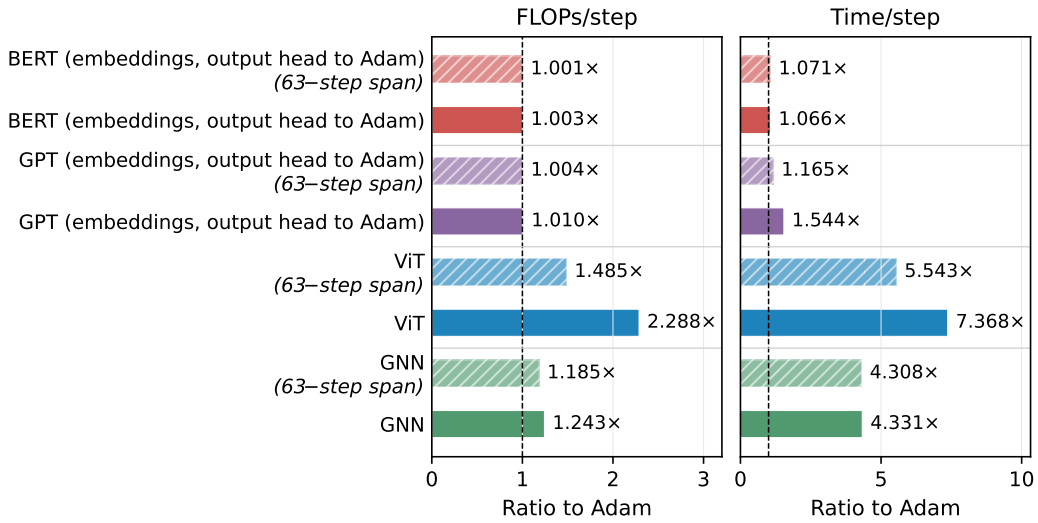

Figure 10: FLOPs and time per training step, reported as ratios to Adam, by history span.

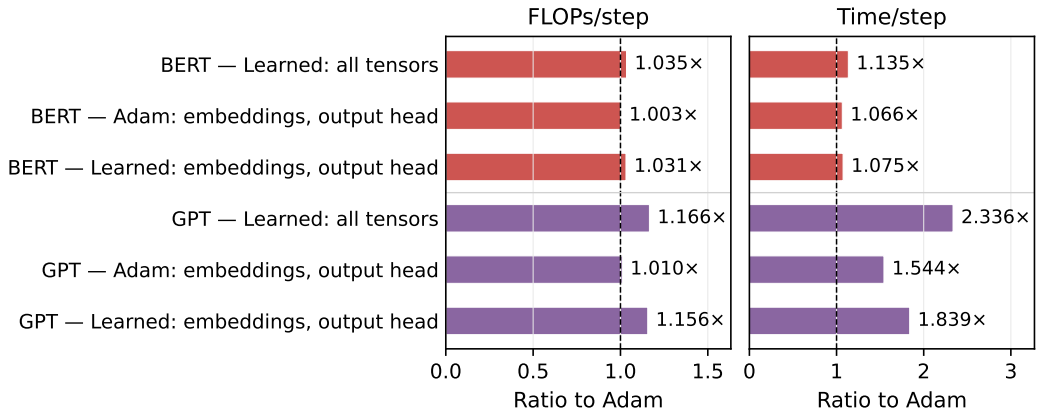

Figure 11: FLOPs and time per training step, reported as ratios to Adam, by language-model parameter assignment.

140, 120, and 60 training nodes for Cora, CiteSeer, and PubMed. Training loss uses the training mask. Full-graph passes retain all nodes and edges. Validation is recorded after every update, and the terminal state is evaluated once on the test mask without model selection.

A.4 Resource measurements

Figures 10 and 11 report deployment cost by history span and language-model parameter assignment, respectively.

The reported 37,389 parameters include trainable policy tensors and exclude fixed positional encodings. Deployment profiles fix model, hardware, precision, batch, parameter assignment, and software configuration. Each repeat uses at least five warm-up and 50 measured updates; the synchronized BERT repeats use 20 warm-up updates. CUDA events measure both the optimizer update and complete training step. The reported ratio is

$$\rho_{\mathrm{FLOP}} = \frac{F_{\mathrm{fb}}^{\mathrm{learned}} + U_{\mathrm{learned}} + U_{\mathrm{Adam,aux}}}{F_{\mathrm{fb}}^{\mathrm{Adam}} + U_{\mathrm{Adam,all}}}, \tag{3}$$

In Equation 3, F_{fb} counts the target forward/backward pass and U counts update arithmetic. Fused Adam kernels omit elementwise arithmetic from the profiler. Both Adam terms use 18 declared

Table 7: Terminal validation loss by language-model parameter assignment. Each row reports the paired Adam baseline. The same trained policy is routed to the listed tensors without separate optimizer training for each assignment. Bold marks the lowest learned-optimizer loss within each model.

Model	Learned optimizer updates	Adam updates	Learned	Adam
BERT-Tiny	All parameters	—	3.889	3.815
	All except embeddings/output head	Embeddings/output head	**3.463**	3.810
	Embeddings/output head	All other parameters	3.484	3.811
GPT-Tiny	All parameters	—	4.299	4.268
	All except embeddings/output head	Embeddings/output head	**4.250**	4.268
	Embeddings/output head	All other parameters	4.309	4.268

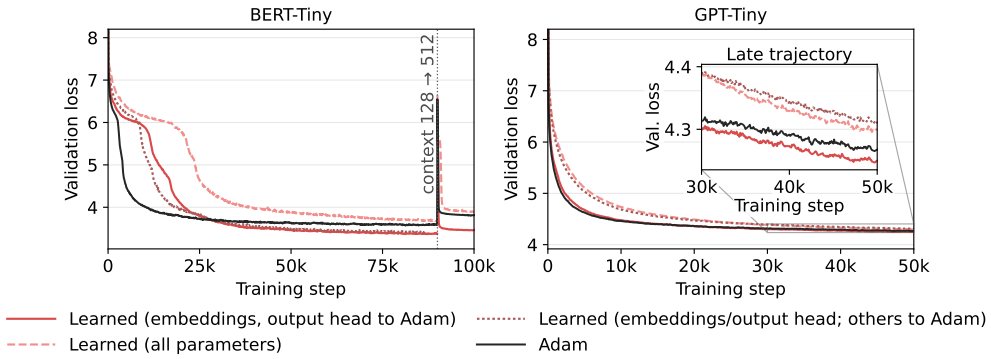

Figure 12: Validation loss when the learned optimizer updates all tensors or either complementary parameter partition. Adam updates the tensors not routed to learned recombination using the same hyperparameters as the all-Adam baseline. BERT changes context at update 90,000, and the GPT inset magnifies updates 30,000–50,000.

scalar operations per updated coordinate. The learned-update term is profiled with Adam disabled before this estimate is added once.

FLOPs and timing are collected separately. Inputs, gradients, and mature history state are resident on-device before measurement. Data loading, validation, logging, and checkpointing are excluded. For language models, Time/step holds each paired Adam forward/backward interval fixed and replaces only its Adam update interval with the measured learned-optimizer interval; this removes between-process target-compute drift from the reported optimizer overhead. Persistent state is measured after history prefill and includes learned-optimizer state, auxiliary Adam state, and policy parameters.

B Additional Evaluation Results and Ablations

B.1 Additional target-model results

B.1.1 Language-model parameter assignments

Table 7 and Figure 12 compare learned recombination on all tensors with the two complementary hybrid assignments. Arms share initialization, minibatches, stochastic streams, and validation panels. The study does not independently train every parameter assignment.

B.1.2 Graph training trajectories

The graph trajectories separate checkpoint selection from compute-normalized optimization. Figure 13 compares the full-span learned optimizer with Adam at equal cumulative FLOPs, while Figure 14 shows the complete validation signal used to select checkpoints without reference to test accuracy.

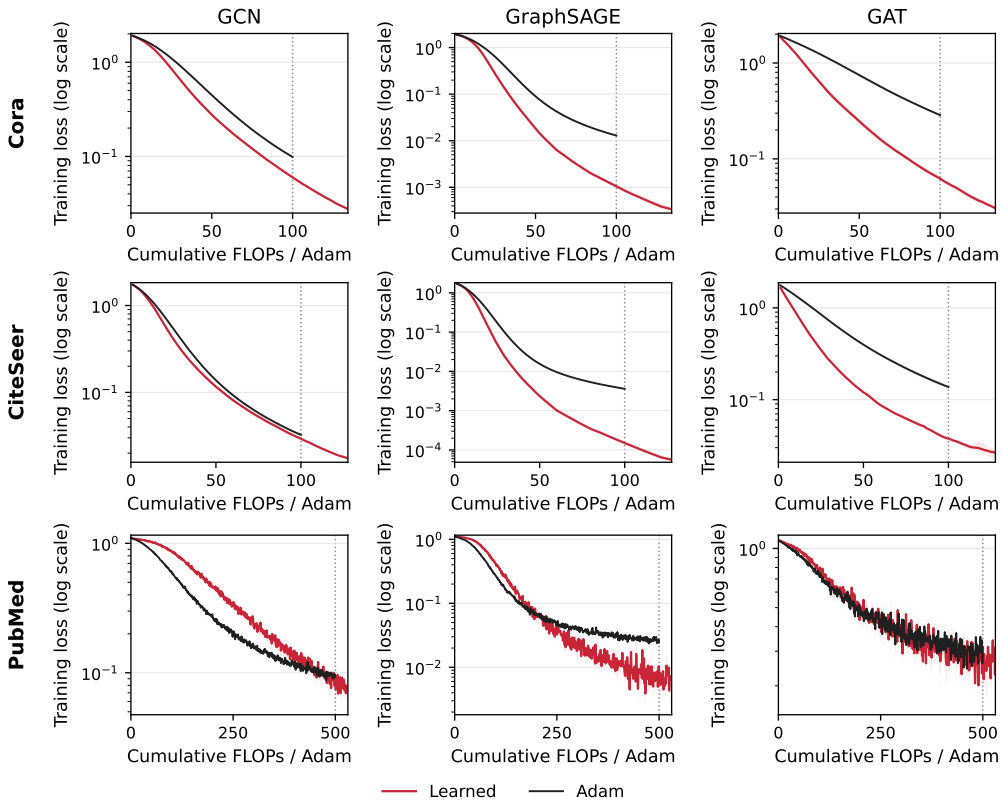

Figure 13: FLOP-aligned GNN training loss for the full-span learned optimizer and Adam. Rows identify datasets, and columns identify model architectures. The horizontal axis measures cumulative profiler-visible FLOPs in units of one Adam step. Lines show three-seed means, and bands show seedwise ranges. Cora and CiteSeer use budgets of 100 Adam steps, while PubMed uses 500 Adam steps. PubMed learned curves continue beyond the dotted 500-step boundary to show the complete 500-update trajectory.

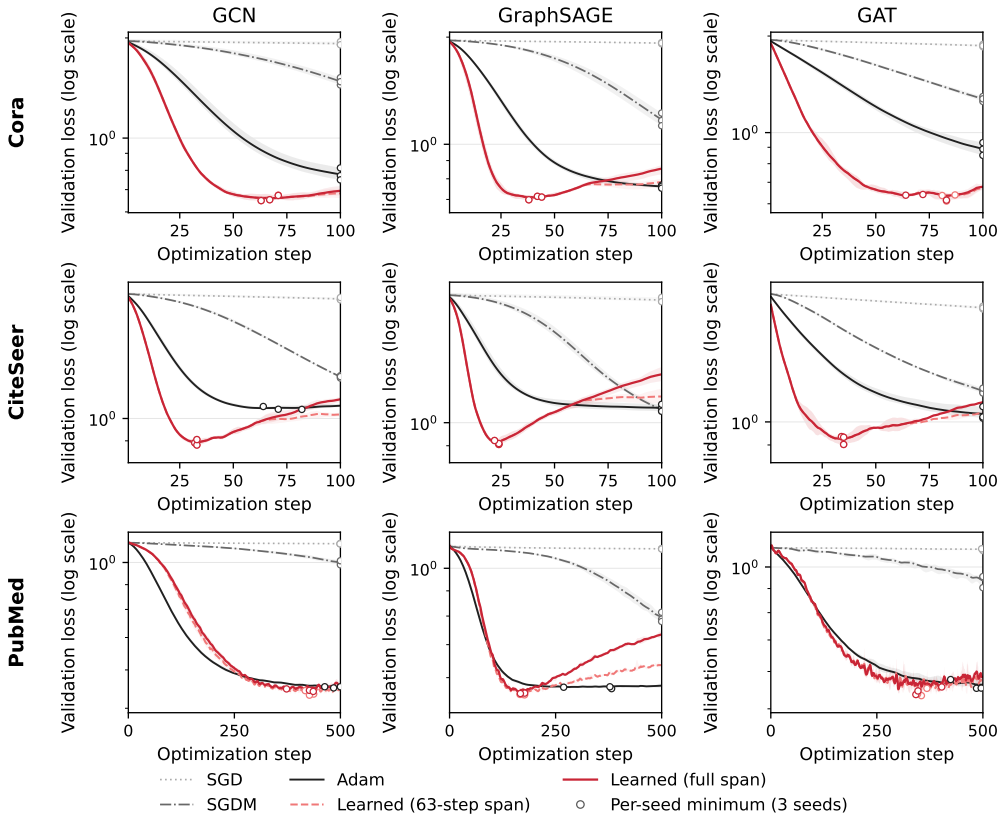

Figure 14: Step-aligned GNN validation loss. Rows identify datasets, and columns identify model architectures. Curves show three-seed means, bands show seedwise ranges, and circles mark the minimum of each seedwise validation trajectory. Checkpoints are selected independently within the fixed 100-update Cora and CiteSeer budgets and 500-update PubMed budget. Test accuracy does not enter checkpoint selection. Figure 13 separately compares training progress at equal cumulative FLOPs.

B.2 Mechanism evidence and ablations

B.2.1 Parameter grouping

Table 8 tests deployment regrouping from the output-slice-trained policy and matched training under alternative groupings. Every trajectory remains finite, but effects vary by task and metric. Matched policies use one optimizer-training seed, and the results do not identify a preferred grouping.

Table 8: Parameter-grouping ablation. Entries are mean ± sample standard deviation over three paired target seeds. The group count is measured at deployment. Output and input slices prioritize complete leading- and second-dimension slices, respectively. Tensor-wide and network-wide conditions form one group per parameter tensor and optimizee. Bold marks the best mean within each task and metric. Adam uses the same target initializations and data streams.

Policy training	Deployment grouping	Deployment groups	Train-loss AUC ↓	Terminal
MicroLeNet/CIFAR-10 (terminal test loss ↓)				
Output slice	Output slice	15	**1.820 ± 0.012**	1.743 ± 0.055
Output slice	Input slice	14	1.823 ± 0.016	1.705 ± 0.054
Output slice	Tensor-wide	8	1.822 ± 0.014	1.681 ± 0.028
Output slice	Network-wide	1	1.831 ± 0.017	1.666 ± 0.018
Input slice	Input slice	14	1.873 ± 0.043	1.748 ± 0.034
Tensor-wide	Tensor-wide	8	1.827 ± 0.024	**1.665 ± 0.034**
Network-wide	Network-wide	1	1.844 ± 0.009	1.693 ± 0.017
Adam	–	–	1.972 ± 0.024	1.804 ± 0.008
GAT/Cora (terminal test accuracy ↑)				
Output slice	Output slice	367	0.195 ± 0.007	80.93 ± 0.15
Output slice	Input slice	367	0.195 ± 0.007	**81.13 ± 0.31**
Output slice	Tensor-wide	8	0.194 ± 0.007	81.00 ± 0.26
Output slice	Network-wide	1	0.194 ± 0.007	81.00 ± 0.10
Input slice	Input slice	367	**0.182 ± 0.007**	80.90 ± 0.10
Tensor-wide	Tensor-wide	8	0.193 ± 0.007	81.00 ± 0.30
Network-wide	Network-wide	1	0.186 ± 0.007	81.00 ± 0.36
Adam	–	–	0.515 ± 0.013	80.37 ± 1.03

B.2.2 History-depth controls

Table 9 compares the 63-step span with the full span over three paired target seeds. Aggregate accuracy is similar, while the preferred span varies across cells. History depth is fixed before validation and test evaluation.

Table 9: GNN history-depth ablation. Entries are test accuracy (%) from the lowest-validation-loss state, reported as mean ± sample standard deviation over three paired target seeds. The final row is the unweighted arithmetic mean of the nine model–dataset means. Bold marks the best mean, and underlining marks the second-best; tied means are both bold.

Data	Model	63-step span (%)	Full span (%)
Cora	GCN	80.77 ± 0.67	**80.83 ± 0.45**
	GraphSAGE	**79.53 ± 0.35**	79.27 ± 0.50
	GAT	80.77 ± 0.71	**80.83 ± 0.51**
CiteSeer	GCN	**70.07 ± 0.35**	**70.07 ± 0.23**
	GraphSAGE	**70.13 ± 0.31**	69.50 ± 0.46
	GAT	**68.07 ± 1.56**	67.60 ± 1.73
PubMed	GCN	**76.43 ± 0.45**	75.77 ± 0.55
	GraphSAGE	**77.77 ± 0.81**	77.13 ± 0.84
	GAT	**75.80 ± 0.53**	75.00 ± 0.87
Arithmetic mean		**75.48 ± 0.21**	75.11 ± 0.23

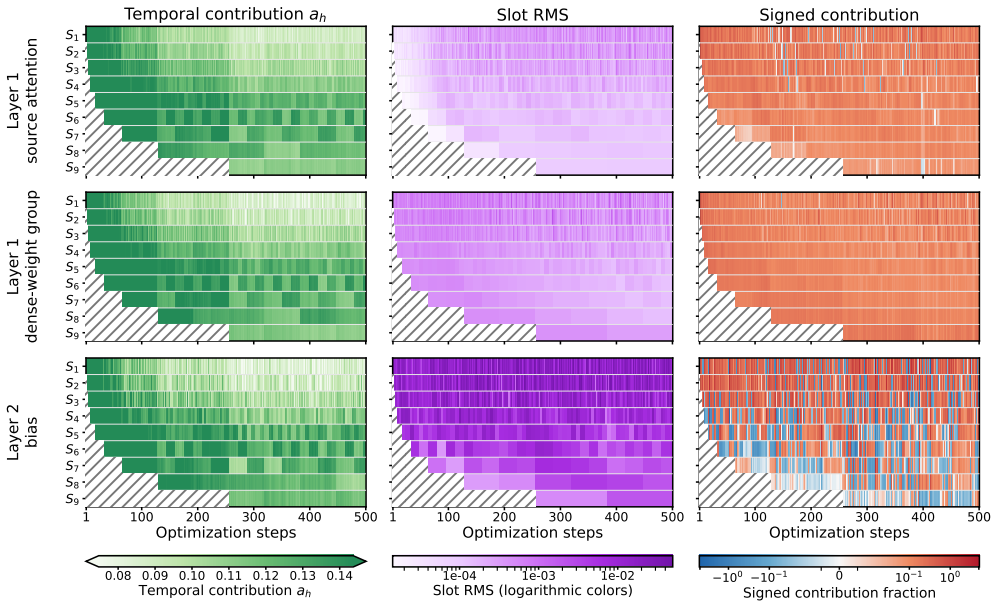

Figure 15: **Full-trajectory groupwise temporal dynamics on GAT/PubMed.** Rows show a 64-coordinate layer-1 source-attention group, a 256-coordinate layer-1 dense-weight group, and the 3-coordinate layer-2 bias. Columns show temporal contributions a_h, raw slot RMS, and signed contribution fractions; the final coefficient is $\alpha_h = sa_h$. RMS colors are logarithmically spaced while their labels retain linear values; signed-fraction colors use a symmetric logarithmic scale. Hatching marks temporal ranges not yet populated.

B.2.3 Temporal coefficients and stored-history content

Figure 15 compares the complete GAT/PubMed trajectories of a source-attention group, a dense-weight group, and the 3-coordinate output bias. Their temporal contributions, slot magnitudes, and signed contributions exhibit distinct patterns along the same model trajectory.

C Spatio-Temporal Extension

This section extends temporal recombination across spatial tiers: multiple partitions of a target-model tensor independently construct candidate updates from gradient history, and a second predictor combines those candidates into the final parameter update.

C.1 Pyramid grouping

Pyramid grouping tests whether the temporal policy can operate over more than one fixed partition of a parameter tensor. Each tensor of rank at least two is represented by its first axis and the flattened product of its remaining axes.

Figure 16 illustrates the generalized spatial hierarchy. The evaluated realization retains the whole-tensor configuration and the two single-axis configurations.

The temporal policy produces one full-tensor candidate $\Delta\theta_t^{(p)}$ from each retained partition p. A separate predictor assigns Softmax weights $\beta_t^{(p)}$ to these candidates and constructs

$$\Delta\theta_t = \sum_p \beta_t^{(p)} \Delta\theta_t^{(p)}. \tag{4}$$

The temporal path applies the same temporal predictor, with shared weights, to every retained partition. The spatial path uses a separate network, also shared across partitions, whose feature encoder, self-attention encoder, and coefficient head follow the temporal architecture but have independently

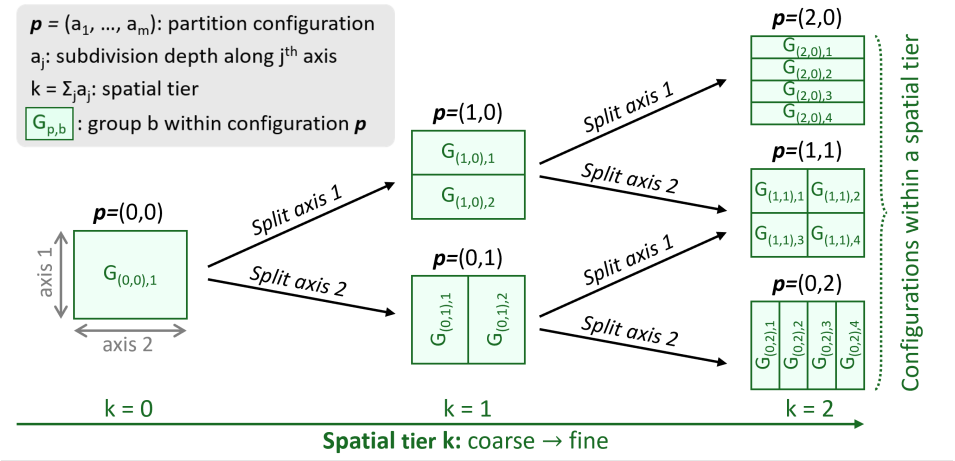

Figure 16: Spatio-temporal extension through pyramid grouping. A partition configuration $p = (a_1, \ldots, a_m)$ assigns subdivision depth a_j to tensor axis j, and spatial tier $k = \sum_j a_j$ contains configurations with the same total subdivision depth. Each cell $G_{p,b}$ denotes parameter group b within configuration p. The illustration uses two tensor axes and split factor $s = 2$ through tier $k = 2$; the evaluated realization retains the whole-tensor and two single-axis configurations with 16 splits per active axis. Within each group, the temporal policy recombines the stored gradient history.

Table 10: Full-trajectory test accuracy (%) for row and pyramid grouping on unseen graph tasks. Cora and CiteSeer use 100 updates and PubMed uses 500. Entries are mean ± sample standard deviation over three target seeds. Bold and underlined values mark the best and second-best mean in each row.

Data	Model	Row	Pyramid	Adam	SGD	SGDM
Cora	GCN	**81.70 ± 0.20**	81.57 ± 0.12	79.27 ± 0.83	31.80 ± 10.01	74.93 ± 0.92
	GraphSAGE	79.60 ± 0.44	**80.00 ± 0.69**	78.73 ± 0.32	24.10 ± 3.58	76.63 ± 0.15
	GAT	81.97 ± 0.31	**82.03 ± 0.50**	78.27 ± 1.26	47.23 ± 8.00	78.97 ± 1.19
CiteSeer	GCN	**71.03 ± 0.81**	70.97 ± 0.67	66.20 ± 1.61	33.00 ± 6.22	70.57 ± 0.71
	GraphSAGE	69.87 ± 0.87	70.13 ± 0.29	65.57 ± 1.36	34.67 ± 9.38	**70.23 ± 0.38**
	GAT	69.00 ± 0.26	69.17 ± 0.32	66.20 ± 0.82	52.87 ± 1.42	**70.37 ± 0.78**
PubMed	GCN	**78.67 ± 0.21**	78.53 ± 0.60	77.50 ± 0.36	51.77 ± 1.15	67.67 ± 6.11
	GraphSAGE	**77.37 ± 0.49**	77.23 ± 0.29	75.17 ± 0.35	50.20 ± 13.67	74.70 ± 0.36
	GAT	78.73 ± 0.31	**78.97 ± 0.12**	77.10 ± 1.42	59.23 ± 5.95	73.30 ± 0.17
Arithmetic mean across 9 cells		76.44 ± 0.28	**76.51 ± 0.16**	73.78 ± 0.26	42.76 ± 2.36	73.04 ± 0.75

learned parameters. The spatial network omits the scale token and group-scale output; its Softmax weights $\beta_t^{(p)}$ directly combine the partition candidates.

The evaluated construction forms three partitions: the whole tensor, 16 contiguous groups along the first axis, and 16 contiguous groups along the flattened trailing axis. Groups smaller than 64 coordinates are omitted. The pyramid policy matches the representative row-grouped policy across nine unseen graph tasks. Table 10 reports test accuracy from the minimum-validation-loss state. Its arithmetic mean is 76.51%, compared with 76.44% for row grouping and 73.78% for Adam. The two learned policies were trained independently, so the comparison establishes that both grouping constructions function under the evaluated protocol rather than isolating the effect of grouping alone.