BOLT: BLOCKWISE RESIDUAL RIDGE SOLVING FOR FROZEN-FEATURE CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Frozen-feature classification repeatedly fits a prediction head on representations produced by a fixed encoder. Although closed-form ridge regression avoids iterative optimization, jointly fitting a wide random-feature head incurs expensive matrix factorizations and large temporary solver memory, while iterative classifier training requires repeated parameter updates. We introduce BOLT (Blockwise Optimizer-free Learning with cyclic refinemenT), a blockwise residual solver for this refitting task. BOLT first decomposes the wide solve into smaller blockwise problems through a sequential residual cascade, enabling a fast one-pass fit. Cyclic refinement then revisits these blocks to correct approximation errors and recover the joint ridge solution, while augmented QR improves numerical stability without forming normal equations. Across five frozen-feature vision benchmarks, BOLT maintains competitive predictive accuracy while substantially reducing peak solver memory compared with the matched joint solve.

1 INTRODUCTION

Frozen encoders turn representation learning into a reusable preprocessing stage: once features are cached, new classifiers can be fitted without updating the backbone (Kornblith et al., 2019; Kolesnikov et al., 2020). This setting is useful when prediction heads must be retrained repeatedly across label sets, data splits, or deployment conditions. Iterative probes remain flexible, but require repeated parameter updates together with learning-rate, stopping, and optimizer choices. Closed-form ridge regression (Hoerl & Kennard, 1970; Sch"olkopf & Smola, 2002) on random features (Huang et al., 2006; Rahimi & Recht, 2007; Le et al., 2013; Rudi & Rosasco, 2017) removes this optimization loop and therefore provides an attractive alternative for repeated refitting.

The closed-form alternative introduces a different bottleneck. A joint ridge solve over a wide random-feature bank requires a large matrix factorization, with leading cost $O(nh^2 + h^3)$ for n samples and total feature width h, and working memory that grows quadratically with the solve width. Reducing the number of random features alleviates this cost but also reduces model capacity. The central challenge is therefore to reduce the scale of the solve without discarding the prescribed feature bank, while retaining a principled connection to the joint ridge solution.

We address this challenge with BOLT, a blockwise residual ridge solver with three complementary components. First, a sequential residual cascade fits small feature blocks to the unexplained target signal, replacing one wide factorization with a sequence of narrower closed-form solves. The resulting one-pass solution preserves the full feature bank but is generally an approximation because earlier blocks are not revised after later blocks are fitted. Second, cyclic refinement revisits every block using the current global residual, correcting this ordering-induced error and converging to the unique joint ridge minimizer under positive regularization. Third, each block is solved through augmented QR rather than normal equations, avoiding the associated squaring of the condition number. Together, these components connect a fast single pass to a numerically stable path toward the matched joint solution.

We evaluate this mechanism on five frozen-feature vision benchmarks against a matched joint ridge solve, block-optimization alternatives, classical random-feature and kernel approximations, and iterative classifier heads. The experiments show that BOLT retains competitive predictive accuracy while reducing peak solver memory relative to the matched joint solve. A separate CIFAR-10 timing

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

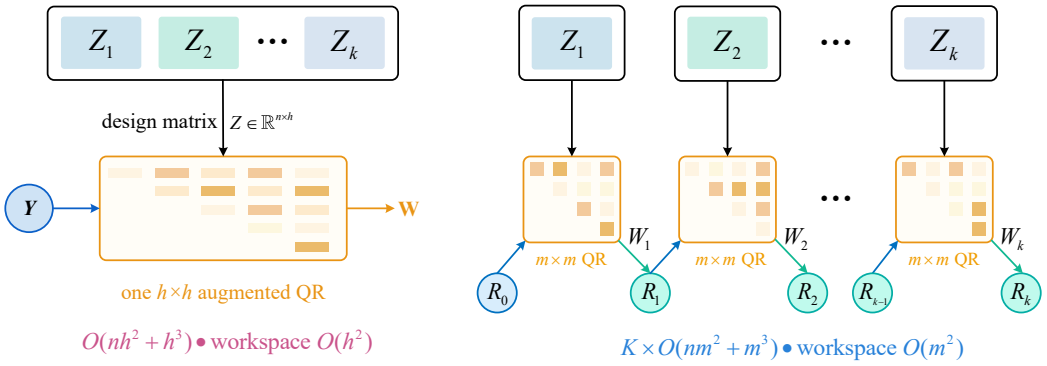


Figure 1: Joint and blockwise ridge solves using the same fixed RBF center bank and total feature width $h = Km$. The joint formulation factors one system in h variables. BOLT instead factors K systems in m variables and transfers information between them through residuals.

study demonstrates faster fitting than the evaluated iterative classifier baselines. Block-count and refinement ablations further characterize how the approximation gap can be exchanged for fitting time and working memory.

The main contributions of this work are threefold:

- We provides competitive predictive accuracy while reducing the peak solver memory and, in favorable cached-feature settings, shortening fitting time relative to the matched joint solve.
- We offers a controllable path from a fast one-pass approximation to the joint ridge solution by adding cyclic refinement when higher accuracy is required.
- Experiments across multiple frozen-feature vision benchmarks demonstrate practical accuracy–time–memory trade-offs together with deterministic closed-form fitting, no gradient-based tuning, and improved numerical stability from augmented QR.

2 Related Work

Random-feature classifiers. Extreme Learning Machines fit a closed-form output layer on randomly generated hidden features (Huang et al., 2006), while Random Fourier Features approximate shift-invariant kernels with randomized finite-dimensional maps (Rahimi & Recht, 2007). Both avoid end-to-end optimization of the feature map, but their standard formulations solve for all output coefficients jointly. BOLT retains the fixed-feature, closed-form setting while organizing the generated features into residual blocks. Its single forward sweep is therefore an approximate solver on a fixed feature bank, rather than a new kernel approximation by itself.

Scalable kernel approximations. Nyström methods reduce kernel cost through a low-rank approximation constructed from sampled columns or landmarks (Williams & Seeger, 2001), while random-feature maps provide an alternative finite-dimensional approximation (Rasmussen & Williams, 2006; Rudi et al., 2017). In contrast, BOLT keeps the prescribed total random-feature width and changes how the corresponding ridge problem is solved. The distinction matters in our comparisons: the matched joint baseline and BOLT use the same feature maps and regularization, whereas the Nyström baseline changes the representation and uses a smaller single-block feature space.

Block optimization. Cyclic block coordinate descent updates one variable block while holding the others fixed and has established convergence guarantees under convexity and regularity conditions (Tseng, 2001; Wright, 2015; Beck & Tetruashvili, 2013; Nesterov, 2012). BOLT uses one Gauss–Seidel pass as a speed-oriented approximation, then optionally continues the same updates as cyclic refinement. We also compare against consensus ADMM (Boyd et al., 2011; Hong et al., 2016);

unlike BOLT's independent coefficient matrix for each feature block, the evaluated consensus formulation constrains blocks through a shared coefficient representation. Thus, its optimization and representational properties differ from those of the matched joint ridge problem.

3 METHOD

3.1 PROBLEM SETUP AND BLOCKWISE CASCADE

Given frozen feature representations $X \in \mathbb{R}^{n\times d}$ from a pre-trained encoder (e.g., ResNet (He et al., 2016), ViT (Dosovitskiy et al., 2021), DINOv2 (Oquab et al., 2024)), we generate K fixed feature blocks $\{\phi_k\}_{k=1}^{K}$, with the feature family chosen by the experiment. The pooling and resource comparisons use standardized RBF features with sampled centers, whereas the refinement and block-configuration experiments use ReLU random projections. Let $Z_k = \phi_k(X) \in \mathbb{R}^{n\times m}$ denote the fixed feature matrix of block k, and let $W_k \in \mathbb{R}^{m\times c}$ be its coefficient matrix for c classes. The joint ridge objective is

$$\mathcal{L}(W_1,\dots,W_K) = \left\| \sum_{k=1}^{K} Z_k W_k - Y \right\|_F^2 + \lambda \sum_{k=1}^{K} \|W_k\|_F^2, \quad \lambda > 0. \tag{1}$$

Let $W = [W_1^\top,\dots,W_K^\top]^\top \in \mathbb{R}^{Km\times c}$ denote the concatenated coefficient matrix. Then $\|W\|_F^2 = \sum_{k=1}^{K} \|W_k\|_F^2$, and Eq. 1 is exactly the objective optimized by the cyclic block updates.

Joint solving minimizes this objective directly but requires $O(n(Km)^2+(Km)^3)$ operations and an $(Km)\times(Km)$ solver workspace. Its cost consequently grows quickly with the total width $h = Km$, motivating solves that operate on width m at a time.

Instead, we build a residual cascade that solves K sequential ridge problems:

$$W_1 = \mathrm{ridge}(Z_1, Y), \tag{2}$$
$$W_k = \mathrm{ridge}(Z_k, R_{k-1}) \quad \text{for } k = 2,\dots,K. \tag{3}$$

where $R_{k-1} = Y - \sum_{j=1}^{k-1} Z_j W_j$ is the residual, and the ridge solver is defined as

$$\mathrm{ridge}(Z_k, R) = (Z_k^\top Z_k + \lambda I)^{-1} Z_k^\top R. \tag{4}$$

When $R = Y - \sum_{j\neq k} Z_j W_j$, this expression is the exact minimizer of $\mathcal{L}$ with respect to W_k while the other blocks are fixed. We evaluate this subproblem using the augmented QR formulation (Appendix B.1). Each block solve costs $O(nm^2 + m^3)$ and uses an $m \times m$ solver workspace, compared with $O(nh^2 + h^3)$ and an $h \times h$ workspace for the joint solve. A complete sweep costs $O(K(nm^2 + m^3))$; it replaces the single cubic term in total width h with K cubic terms in block width m. For c targets, constructing the right-hand sides and coefficients adds $O(K(nmc + m^2c))$.

3.2 SINGLE-SWEEP APPROXIMATION

The forward cascade is a Gauss–Seidel pass initialized at $W_1 = \cdots = W_K = 0$. At block k, it solves

$$W_k^{\mathrm{fwd}} = \arg\min_{U} \left\| Z_k U - \left(Y - \sum_{j<k} Z_j W_j^{\mathrm{fwd}} \right) \right\|_F^2 + \lambda \|U\|_F^2. \tag{5}$$

Earlier coefficients are not revisited during this pass. Consequently, the forward solution generally differs from the joint optimum when feature blocks are correlated: an early block can explain a component that a later block would represent more efficiently, but the later solve cannot revise that earlier allocation. This is the source of the single-sweep approximation gap. It is not a reduction in feature capacity, because BOLT and the matched joint baseline use the same concatenated feature bank.

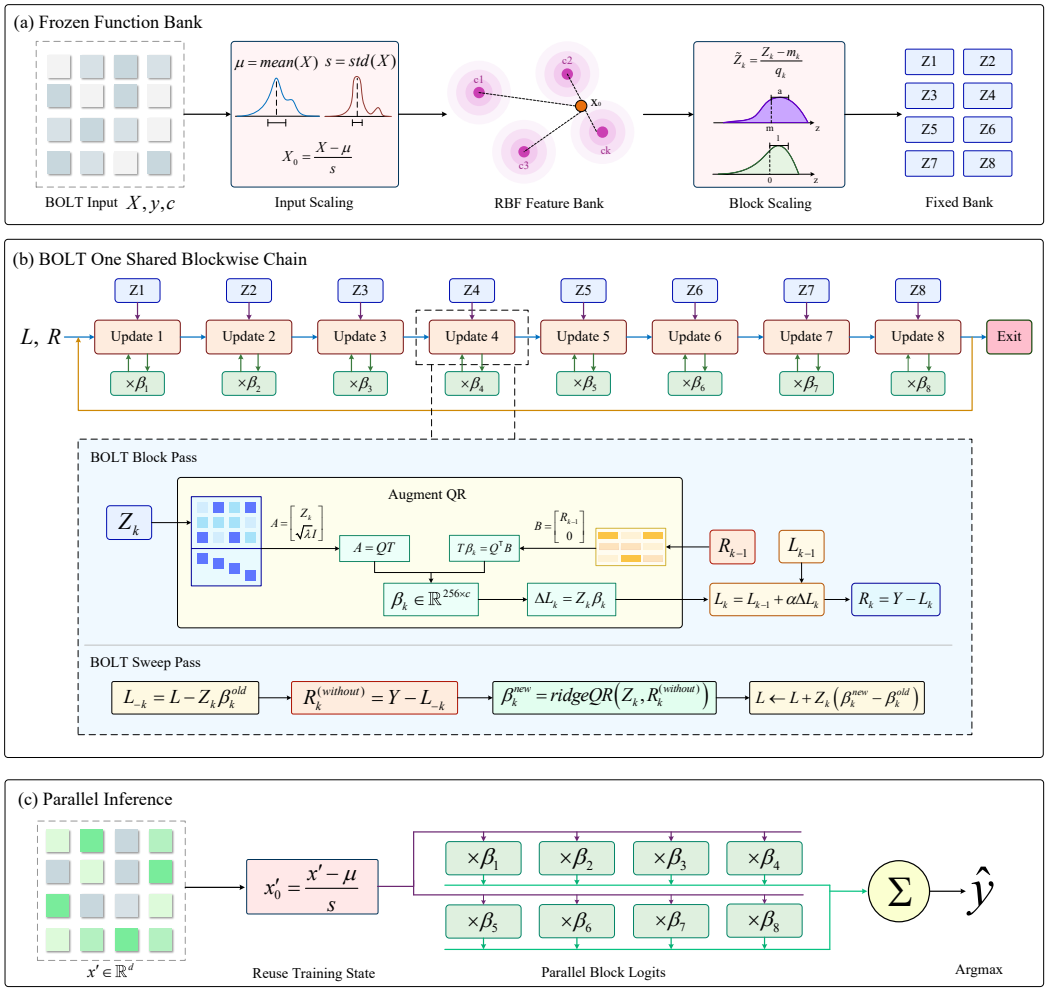


Figure 2: Overview of BOLT training and inference. Each block has its own fixed random feature map with a common block width; blocks do not share parameters and are coupled only through the residual updates. During a forward sweep, blocks are fitted in residual order; optional cyclic refinement revisits the same fixed blocks. At inference, block contributions are aggregated into the final prediction.

Block order can therefore affect a single sweep. Our main implementation fixes the sampled maps and their order for reproducibility; alternating or repeated sweeps reduce sensitivity to the initial order by allowing every block to respond to the current contributions of all other blocks. The empirical block-count ablation in Section 5.4 measures the resulting accuracy–time trade-off rather than assuming that more blocks are always preferable.

3.3 Cyclic Refinement and Inference

After the initial forward sweep, we revisit blocks in sequence, re-solving each on the current global residual:

The complete training procedure is given in Algorithm 1 in Appendix B; the main text focuses on its optimization interpretation and empirical consequences.

Here $\alpha \in (0, 1]$ is a damping factor that blends old and new coefficients: $W_k \leftarrow (1 - \alpha)W_k^{\text{old}} + \alpha \cdot W_k^{\text{new}}$. The update $W_k^{(s+1)} = (1 - \alpha)W_k^{(s)} + \alpha \cdot \tilde{W}_k^{(s+1)}$ is a convex combination of the current iterate and the exact block minimizer. By convexity, this update does not increase the objective.

Under the standard assumptions for cyclic block coordinate descent on strongly convex objectives, repeated updates converge to the unique minimizer of the ridge objective (Bertsekas, 1999; Wright, 2015). We use $\alpha = 0.8$ in the refinement experiments; the tested range is reported in Appendix B.5.

Computational cost per sweep: $O(K \cdot (nm^2 + m^3))$ for forming augmented systems and computing QR factorizations, plus $O(K \cdot (nmc + m^2c))$ for c class targets. Total cost scales as $O(S \cdot K \cdot (nm^2 + m^3 + nmc + m^2c))$ for S sweeps. Experimental configuration: $K = 6$ blocks of width $m = 192$ (1,152 total features) for reproducibility. In the seeded ablation in Section 5.4, $K = 1$ gives the highest accuracy, while $K = 6$ offers a favorable speed–accuracy trade-off for this setting.

The residual sequence is required only while fitting the coefficients. At inference, the stored feature maps and standardization statistics produce $Z_k(x)$ independently, and the classifier aggregates their logits as $\hat{y}(x) = \sum_k Z_k(x)W_k$. Thus, cyclic refinement changes the learned coefficients but not the inference rule. The block contributions may be computed independently when hardware permits, followed by a sum over the class logits.

4 Theoretical Analysis

4.1 Block-Coordinate Interpretation

Let $Z = [Z_1, \ldots, Z_K]$ and define $H_k = Z_k^\top Z_k + \lambda I$. Because $\lambda > 0$, every H_k is positive definite and every block subproblem has a unique minimizer. Moreover, the Hessian of the full objective is $2(Z^\top Z + \lambda I) \succ 0$, so the joint ridge objective has a unique global minimizer. A forward BOLT sweep is therefore the first Gauss–Seidel cycle of block coordinate descent from the zero initialization; cyclic refinement simply continues these cycles.

This view separates two properties. The single sweep is a finite-computation approximation and need not equal the joint solution. In contrast, the limiting point of exact cyclic updates is the joint solution itself. Damping changes the path to this limit, but not the fixed point: if every block equals its conditional minimizer, the concatenated first-order condition is $(Z^\top Z + \lambda I)W = Z^\top Y$.

4.2 Descent and Convergence

Theorem 1 (Convergence of Cyclic Block Refinement)**.** *Let $W_k^{(s)}$ denote the coefficient matrix of block k after the s-th complete cyclic sweep. Assume $\lambda > 0$, fixed feature blocks $\{Z_k\}_{k=1}^K$, exact solution of each block ridge subproblem, cyclic visitation of all blocks, and damping parameter $\alpha \in (0, 1]$. Let $\mathcal{L}(W_1, \ldots, W_K)$ denote the joint ridge objective in Eq. 1. Then:*

(1) ***Monotonic decrease****: The objective value is nonincreasing after every block update.*

(2) ***Convergence****: The sequence $(W_1^{(s)}, \ldots, W_K^{(s)})$ converges to the unique minimizer of $\mathcal{L}$ as $s \to \infty$.*

Proof sketch: Fix all blocks except k, let $\widetilde{W}_k$ be the exact block minimizer, and set $D_k = \widetilde{W}_k - W_k$. The damped update is

$$W_k^+ = W_k + \alpha D_k. \tag{6}$$

Using the first-order condition at $\widetilde{W}_k$ and expanding the quadratic gives the exact change

$$\mathcal{L}(W_k^+) - \mathcal{L}(W_k) = -\alpha(2 - \alpha)\,\mathrm{tr}(D_k^\top H_k D_k) \leq 0. \tag{7}$$

Equality holds only when the block is already optimal. The objective is bounded below, every block is visited cyclically, and the full objective is strongly convex; standard cyclic BCD results then give convergence to its unique minimizer (Tseng, 2001; Wright, 2015). Appendix A.1 provides the full argument.

Figure 3(b) illustrates the corresponding empirical behavior on CIFAR-100-ViT. For the six-block configuration, the test-accuracy gap to the matched joint solution narrows with additional sweeps, although test accuracy is not itself the optimized objective and need not decrease monotonically by theorem.

4.3 Complexity and Working Memory

For $h = Km$, a joint QR-based ridge solve has leading arithmetic cost $O(nh^2 + h^3)$ and solver workspace quadratic in h. One BOLT sweep performs K width-m solves, with cost $O(K(nm^2 + m^3))$ plus $O(K(nmc + m^2c))$ for c right-hand sides. Holding h fixed and substituting $m = h/K$ reduces the leading factorization terms to $O(nh^2/K + h^3/K^2)$. More directly, the ratio of these leading terms is

$$\rho_K \;=\; \frac{nh^2 + h^3}{nh^2/K + h^3/K^2} \;=\; K\,\frac{n+h}{n+h/K}. \tag{8}$$

Thus, when the sample count dominates the feature width, one sweep reduces the leading factorization work by approximately $K\times$; when the cubic factorization term dominates, the reduction approaches $K^2\times$. For example, with $K = 6$, $n = 50{,}000$, and $h = 1{,}152$, the leading-term estimate is about $6.1\times$. This is an arithmetic estimate rather than a wall-clock guarantee: residual updates, feature construction, memory traffic, and repeated kernel launches can make measured speedups smaller.

Sequential processing requires only one block design matrix and one block solver workspace at a time, giving $O(nm+m^2)$ peak working storage beyond the cached inputs, rather than $O(nh+h^2)$ for a materialized joint solve. The learned model still stores all K feature maps and coefficient blocks, so blockwise solving reduces temporary solver memory rather than the final parameter count.

4.4 Numerical Stability of the Ridge Solve

The inverse in Eq. 4 defines the exact block minimizer but is not formed explicitly. We solve the augmented least-squares system

$$\widetilde{Z}_k = \begin{bmatrix} Z_k \\ \sqrt{\lambda} I \end{bmatrix}, \qquad \widetilde{R} = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{9}$$

by QR factorization. This avoids explicitly forming the normal-equation matrix, which squares the condition number relevant to the unregularized least-squares calculation (Bj"orck, 1996; Golub & Van Loan, 2013).

Proposition 1 (Conditioning of augmented ridge regression)**.** *Let $\bar{\sigma}_1 \geq \cdots \geq \bar{\sigma}_m \geq 0$ denote the singular values of Z_k, padded with zeros when necessary. For $\lambda > 0$,*

$$\kappa_2(\widetilde{Z}_k) = \sqrt{\frac{\bar{\sigma}_1^2 + \lambda}{\bar{\sigma}_m^2 + \lambda}} = \sqrt{\kappa_2(Z_k^\top Z_k + \lambda I)}. \tag{10}$$

Thus augmented QR operates on a system whose condition number is the square root of that of the regularized normal-equation matrix. The proof is given in Appendix A.2.

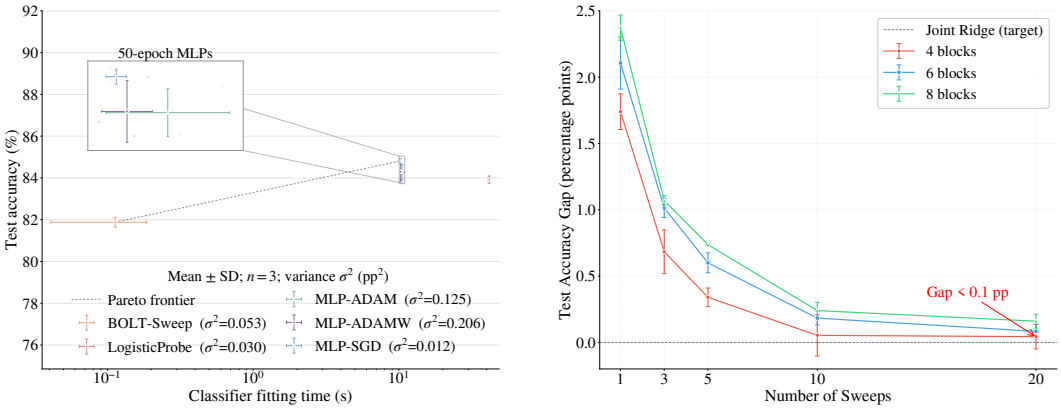
Table 1: Cross-dataset comparison of BOLT-Block and the matched joint ridge solve. Values are means over 3 seeds; time is fitting time in seconds and memory is peak allocated GPU memory in MB.

Dataset	BOLT Acc.	Joint Acc.	BOLT Time	Joint Time	Peak MB (B/J)
CIFAR-100-R18	63.35 ± 0.12	67.28 ± 0.14	0.156 ± 0.002	0.179 ± 0.009	433/1273
CIFAR-100-ViT	76.74 ± 0.17	79.41 ± 0.21	0.164 ± 0.001	0.182 ± 0.001	480/1319
flowers102-DINOv2	99.56 ± 0.02	99.58 ± 0.01	0.017 ± 0.001	0.027 ± 0.009	28/99

5 Experiments

5.1 Experimental Setup

Datasets: MNIST-HOG (10 classes, 1,296-dim HOG), CIFAR-10-ResNet18 (10 classes, 512-dim), CIFAR-100-ResNet18 (100 classes, 512-dim), CIFAR-100-ViT (100 classes, 768-dim ViT-B/16), flowers102 (102 classes, 384-dim DINOv2-small). This spans handcrafted vs learned features, CNN



(a) Accuracy–time trade-off on CIFAR-10

(b) Refinement convergence on CIFAR-100-ViT

Figure 3: Efficiency and convergence results on cached features. (a) Speed–accuracy trade-off against iterative classifier baselines on CIFAR-10-ResNet18 under the same timing protocol. (b) Test-accuracy gap to the matched joint ridge solution on CIFAR-100-ViT for six blocks of width 192; the mean gap decreases from 2.11 percentage points after one sweep to 0.08 after 20 sweeps.

vs Transformer vs self-supervised architectures (Krizhevsky, 2009; Caron et al., 2021; Chen et al., 2020; Grill et al., 2020), 10- vs 100-class problems, and large-sample (50k) vs small-sample (1k) regimes.

Baselines: Joint-RBF-QR (matched joint-solve reference), Block Coordinate Descent (BCD-Classic, validates convergence), ADMM-Consensus, Random Fourier Features (RFF), Nyström Approximation, Extreme Learning Machine (ELM), Logistic Regression Probe, and MLPs with SGD/Adam/AdamW/RMSprop (Kingma & Ba, 2015; Loshchilov & Hutter, 2019). Implementation: PyTorch 2.0, CUDA 11.8, NVIDIA RTX 3090. Timings averaged over 3 seeds with GPU synchronization. Full configuration in Appendix B.

5.2 Main Results: Baseline Comparison

We first compare BOLT-Block with the matched joint ridge solve on three representative frozen-feature settings (Table 1), spanning a large-sample CNN representation, a transformer representation, and a small-sample self-supervised representation.

Key findings: Across these three representative settings, BOLT-Block retains the same frozen feature bank and remains close to the matched joint accuracy, while reducing peak solver memory by 63.6–71.5%. Its fitting-time advantage is strongest in the small-sample flowers102 setting; the broader iterative-classifier timing comparison appears in Figure 3(a).

Figure 3(a) visualizes the CIFAR-10 speed-accuracy Pareto frontier, showing our method lies on the empirical Pareto frontier among the baselines evaluated under this timing protocol. Figure 3(b) then analyzes cyclic refinement on the more challenging CIFAR-100-ViT setting used in the block ablations.

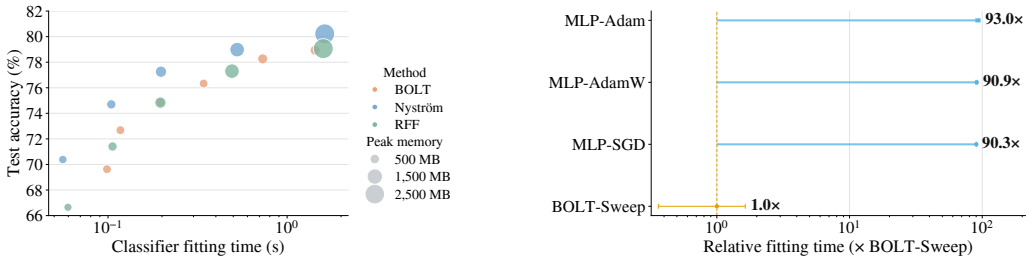
5.3 Multi-Dataset Pooling Comparison

Table 2 compares 8 pooling methods across 5 datasets, including the matched joint-ridge reference (Joint-RBF-QR), the iterative convergence baseline (BCD-Classic), and our blockwise methods. This batch uses the same standardized RBF center bank, with $K = 8$ blocks of $m = 256$ features per block and total width 2,048. BOLT-Sweep refers to three alternating sweeps through the blocks; BOLT-Block refers to the residual cascade evaluated in a single forward pass.

BOLT-Block achieves 83.94% mean accuracy, 1.79 percentage points below the matched joint solve. Fit-time comparisons are dataset-dependent: on flowers102, BOLT-Block trains in 17.3ms versus 27.3ms for Joint-RBF-QR (1.57× speedup), while its fit time is slightly higher than the joint solve on

Table 2: Mean test accuracy (%) across pooling methods over 3 seeds. Multi-dataset experiments use standardized RBF features with total width 2048 (block width 256). The matched joint-ridge baseline uses the same fixed center bank, total feature width, regularization coefficient, and label encoding as BOLT.

Method	MNIST-HOG	CIFAR-10-ResNet18	CIFAR-100-ResNet18	CIFAR-100-ViT	flowers102-DINOv2	Mean
Joint-RBF-QR	98.22 ± 0.06	84.16 ± 0.08	67.28 ± 0.14	79.41 ± 0.21	**99.58 ± 0.01**	**85.73 ± 0.09**
BCD-Classic	97.80 ± 0.02	83.31 ± 0.14	65.11 ± 0.09	78.04 ± 0.13	**99.58 ± 0.01**	84.77 ± 0.02
BOLT-Sweep	97.47 ± 0.02	82.84 ± 0.16	63.80 ± 0.12	76.96 ± 0.06	99.57 ± 0.01	84.13 ± 0.06
BOLT-Block	97.41 ± 0.06	82.61 ± 0.16	63.35 ± 0.12	76.74 ± 0.17	99.56 ± 0.02	83.94 ± 0.02
RFF	96.37 ± 0.23	80.27 ± 0.57	63.56 ± 0.18	76.52 ± 0.11	99.13 ± 0.07	83.17 ± 0.15
Parallel-Fusion	96.58 ± 0.07	81.07 ± 0.23	60.04 ± 0.23	73.05 ± 0.12	99.47 ± 0.01	82.04 ± 0.10
Nyström	96.16 ± 0.24	80.17 ± 0.06	57.32 ± 0.36	70.67 ± 0.18	99.28 ± 0.10	80.72 ± 0.12
ADMM-Consensus	89.55 ± 0.63	72.04 ± 1.11	41.49 ± 0.91	58.42 ± 1.00	97.92 ± 0.18	71.88 ± 0.51



(a) Accuracy–time–memory trade-off

(b) Fitting-speed comparison

Figure 4: Resource and fitting-speed comparisons on cached features. (a) Accuracy–time–memory trade-off among BOLT, Nyström, and random Fourier feature (RFF) methods on CIFAR-100-ViT, with marker size indicating peak memory. (b) Relative fitting time on CIFAR-10-ResNet18, normalized by BOLT-Sweep, for BOLT-Sweep and the evaluated MLP baselines.

the other four datasets. On flowers102 it also reduces peak memory from 98.8MB to 28.2MB (71.5% reduction) with only 0.02 percentage points accuracy loss. BCD-Classic reaches 84.77% mean accuracy (0.96 percentage points below the joint solve). Detailed analysis appears in Appendix C.

Figure 4 complements Table 1 with two resource-oriented views. Figure 4(a) shows the accuracy–time–memory trade-off among BOLT, Nyström, and RFF on CIFAR-100-ViT cached features. Figure 4(b) reports the separate CIFAR-10-ResNet18 timing comparison against the evaluated MLP baselines, normalized by BOLT-Sweep.

5.4 Block Configuration Analysis

Table 3 systematically tests block configurations on CIFAR-100-ViT with fixed capacity (1,152 features).

Note: Tables 3 and 4 report means and sample standard deviations over three independently seeded random feature banks. The QR solve is deterministic conditional on each sampled feature bank; the reported variation therefore reflects feature-bank randomness rather than solver noise.

The multi-dataset comparison (Table 2) uses total width 2,048 for broader coverage, while convergence experiments and this ablation use 1,152 features for consistency with the initial configuration.

Key findings: For the evaluated CIFAR-100-ViT ablation with a fixed total width of 1,152 features, one block gives the highest observed accuracy (76.59%), while six blocks provide a favorable speed–accuracy trade-off (74.69% in 0.917s, 1.63× speedup). Increasing the block count beyond six gives only small additional speed gains and slightly lower accuracy. This result should not be interpreted as a universal optimum for all datasets or feature widths.

Table 4 decomposes the accuracy gap via cyclic refinement.

Table 3: Block configuration ablation on CIFAR-100-ViT. Fixed capacity 1,152 features is distributed across varying blocks. Accuracy values are means ± sample standard deviations and times are means over three independently seeded feature banks.

Blocks	Block Width	Accuracy (%)	Time (s)	Speedup
1	1152	**76.59 ± 0.19**	1.493 ± 0.006	1.00×
2	576	75.83 ± 0.14	1.047 ± 0.002	1.43×
3	384	75.10 ± 0.10	1.029 ± 0.012	1.45×
4	288	74.85 ± 0.10	0.965 ± 0.018	1.55×
6	**192**	**74.69 ± 0.15**	**0.917 ± 0.004**	**1.63×**
8	144	74.27 ± 0.07	0.871 ± 0.029	1.71×
12	96	74.07 ± 0.31	0.930 ± 0.024	1.61×

Table 4: Accuracy gap decomposition on CIFAR-100-ViT (6 blocks × 192). Accuracy values are means ± sample standard deviations and times are means over three independently seeded feature banks. The matched joint-QR baseline uses the same six standardized feature blocks and achieves $76.80 \pm 0.16\%$.

Configuration	Sweeps	Accuracy (%)	Time (s)
One-sweep blockwise	1	74.69 ± 0.15	0.945 ± 0.012
5-sweep blockwise	5	76.20 ± 0.19	4.029 ± 0.026
10-sweep blockwise	**10**	**76.62 ± 0.13**	**7.800 ± 0.066**
20-sweep blockwise	20	76.72 ± 0.11	15.947 ± 0.036
ELM-1152	1	76.63 ± 0.09	1.224 ± 0.015

Gap breakdown (one-sweep 74.69% vs matched joint QR 76.80%): the total gap is 2.11 percentage points. Twenty-sweep refinement closes 2.02 percentage points, corresponding to 96.1% of the initial gap, and leaves a mean gap of 0.08 percentage points. This empirically supports the predicted approach to the joint ridge solution. In the CIFAR-10 convergence study, the 20-sweep gap is below 0.1 percentage points only for the 4-block configuration; the 6- and 8-block configurations retain gaps of 0.21 and 0.13 percentage points, respectively (Table 5). For comparison, ELM-1152 achieves 76.63% in 1.224s; the matched-joint result here is a separate same-feature reference.

6 Conclusion

We introduced BOLT (Blockwise Optimizer-free Learning with cyclic refinemenT), a blockwise ridge cascade for frozen-feature classification. Its block solves replace the joint cubic term in the total feature width $h = Km$ with cubic terms in the block width m. The method avoids learning-rate schedules, early stopping, and gradient descent; measured wall-clock gains are dataset- and configuration-dependent. Our augmented QR solver improves numerical stability, and cyclic refinement provably converges to the joint ridge solution. On CIFAR-100-ViT, ten cyclic sweeps reduce the mean gap to the matched joint ridge baseline to 0.18 percentage points, while twenty sweeps reduce it to 0.08 percentage points. In the CIFAR-10 convergence experiment, only the 4-block configuration falls below a 0.1 percentage-point gap after 20 sweeps. The results support BOLT as a practical option for repeated fitting of cached frozen features when its memory and accuracy trade-offs are acceptable.

AI Use Statement

In this work, we used generative AI tools for manuscript language polishing, formula typesetting, and figure-caption refinement. We did not use generative AI tools for developing the core methodology, theoretical or mathematical claims, experimental design or implementation, or result analysis

and interpretation; other research tasks requiring disclosure, such as synthetic-data generation and qualitative-data analysis, are not applicable to this work. We also used generative AI tools for preliminary literature-search assistance. We reviewed all AI-assisted work and manually cross-checked AI-polished text against our original arguments. All experiment-related descriptions were independently verified by two authors. We take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Amir Beck and Luba Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013. doi: 10.1137/120887679.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms toward AI. In *Large-Scale Kernel Machines*, pp. 321–360. MIT Press, 2007.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 1999.

Åke Bj"orck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186, 2010.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9650–9660, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 1597–1607, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

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

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4 edition, 2013.

Gene H. Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634.

Mingyi Hong, Zhi-Quan Luo, and Meisam Razaviyayn. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization*, 26(1):337–364, 2016.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1–3):489–501, 2006.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European Conference on Computer Vision*, pp. 491–507, 2020.

Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2661–2671, 2019.

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

Quoc V. Le, Tamas Sarlós, and Alexander J. Smola. Fastfood - computing hilbert space expansions in loglinear time. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 244–252, 2013.

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.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2 edition, 2006.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Mike Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Alessandro Rudi and Lorenzo Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

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

Bernhard Sch"olkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10347–10357, 2021.

Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, volume 13, pp. 682–688, 2001.

Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

A Theoretical Proofs

A.1 Proof of Theorem 1: Convergence to Joint Ridge Solution

Proof. Let $Z = [Z_1, \ldots, Z_K]$ and $W = [W_1^\top, \ldots, W_K^\top]^\top$. The Hessian of $\mathcal{L}(W) = \|ZW - Y\|_F^2 + \lambda\|W\|_F^2$ is $2(Z^\top Z + \lambda I) \succ 0$, so $\mathcal{L}$ is strongly convex and coercive and has a unique minimizer W^*.

Consider one update of block k with all other blocks fixed. Write $H_k = Z_k^\top Z_k + \lambda I \succ 0$, let $\widetilde{W}_k$ be the exact block minimizer, and set $D_k = \widetilde{W}_k - W_k$. The block gradient at W_k is $-2H_kD_k$. Expanding the quadratic at $W_k^+ = W_k + \alpha D_k$ gives

$$\mathcal{L}(W_k^+) - \mathcal{L}(W_k) = \alpha\langle -2H_kD_k, D_k\rangle_F + \alpha^2 \operatorname{tr}(D_k^\top H_k D_k) \tag{11}$$

$$= -\alpha(2-\alpha)\operatorname{tr}(D_k^\top H_k D_k). \tag{12}$$

For $\alpha \in (0, 1]$, this quantity is nonpositive and is zero only if $D_k = 0$. Hence every block update decreases the objective unless that block is already conditionally optimal.

The iterates remain in the compact initial level set because the objective is coercive and nonincreasing. Summing the decreases shows that the block displacements tend to zero. Since every block is visited cyclically and each block minimizer is unique and continuous in the remaining blocks, every accumulation point is conditionally optimal in every block. Its full gradient is therefore zero. Strong convexity implies that the only such point is W^*, so all accumulation points coincide and the full sequence converges to W^*. This is also a direct specialization of standard cyclic block-coordinate convergence results (Tseng, 2001; Wright, 2015). □

A.2 Proof of Proposition 1: Numerical Stability

Proof. Let $r = \min(n, m)$ and let $\sigma_1 \geq \ldots \geq \sigma_r \geq 0$ be the singular values of $X \in \mathbb{R}^{n\times m}$. When $m > r$, extend the singular value sequence with zeros: $\bar{\sigma}_{r+1} = \cdots = \bar{\sigma}_m = 0$, where $\bar{\sigma}_i$ denotes the extended sequence. With $\lambda > 0$, the augmented matrix $\tilde{X} = [X; \sqrt{\lambda}I]$ has full column rank.

General case: Form $A = X^\top X + \lambda I$ and solve $AW = X^\top Y$. The matrix $X^\top X$ has eigenvalues $\{\bar{\sigma}_i^2\}_{i=1}^m$ (including zeros when $m > r$), so A has eigenvalues $\bar{\sigma}_i^2 + \lambda$ for $i = 1, \ldots, m$. The 2-norm condition number satisfies:

$$\kappa_2(X^\top X + \lambda I) = \frac{\bar{\sigma}_1^2 + \lambda}{\bar{\sigma}_m^2 + \lambda} \tag{13}$$

Stack $\tilde{X} = [X; \sqrt{\lambda}I] \in \mathbb{R}^{(n+m)\times m}$ and $\tilde{Y} = [Y; 0]$. The matrix $\tilde{X}^\top\tilde{X} = X^\top X + \lambda I$, which has eigenvalues $\bar{\sigma}_i^2 + \lambda$ for $i = 1, \ldots, m$. Therefore, the singular values $\tilde{\sigma}_i$ of $\tilde{X}$ satisfy:

$$\tilde{\sigma}_i^2 = \bar{\sigma}_i^2 + \lambda \quad \text{for } i = 1, \ldots, m \tag{14}$$

Therefore:

$$\kappa_2(\tilde{X}) = \frac{\tilde{\sigma}_1}{\tilde{\sigma}_m} = \frac{\sqrt{\bar{\sigma}_1^2 + \lambda}}{\sqrt{\bar{\sigma}_m^2 + \lambda}} = \sqrt{\kappa_2(X^\top X + \lambda I)} \tag{15}$$

This establishes $\kappa_2(\tilde{X})^2 = \kappa_2(X^\top X + \lambda I)$. The augmented QR approach works directly with $\tilde{X}$ (condition number $\kappa_2(\tilde{X})$) rather than forming the Gram matrix (condition number $\kappa_2(X^\top X + \lambda I) = \kappa_2(\tilde{X})^2$), avoiding the squaring of singular values and improving numerical stability under ill-conditioning.

Full-rank example: If X has full column rank with $\kappa_2(X) = \sigma_1/\sigma_m = 10^6$ and $\lambda \ll \sigma_m^2$, then $\kappa_2(X^\top X + \lambda I) \approx \sigma_1^2/\sigma_m^2 = 10^{12}$ (normal equations) versus $\kappa_2(\tilde{X}) \approx \sigma_1/\sigma_m = 10^6$ (augmented QR)—six orders of magnitude more stable.

Rank-deficient example: If $m > r$ (e.g., $m = 1000$, $r = 500$), then $\bar{\sigma}_m = 0$. With $\lambda = 0.01$ and $\bar{\sigma}_1 = 100$, we have $\kappa_2(X^\top X + \lambda I) = (\bar{\sigma}_1^2 + \lambda)/\lambda = (100^2 + 0.01)/0.01 \approx 10^6$ (normal equations) versus $\kappa_2(\tilde{X}) = \sqrt{(\bar{\sigma}_1^2 + \lambda)/\lambda} = \sqrt{(100^2 + 0.01)/0.01} \approx 10^3$ (augmented QR)—three orders of magnitude more stable. □

B Implementation Details

B.1 Augmented QR Ridge Solver

Standard ridge regression via normal equations $(X^\top X + \lambda I)^{-1}X^\top Y$ suffers from numerical instability when $X^\top X$ is ill-conditioned. We use the augmented QR approach:

$$\tilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}, \quad \tilde{Y} = \begin{bmatrix} Y \\ 0 \end{bmatrix} \tag{16}$$

Performing QR decomposition $\tilde{X} = QR$ and solving $RW = Q^\top \tilde{Y}$ via back-substitution yields the ridge solution without forming $X^\top X$. This approach: (1) avoids squaring the condition number, (2) naturally incorporates regularization through stacking, and (3) enables reuse of Q and R for multiple right-hand sides.

B.2 Blockwise Ridge with Cyclic Refinement

Algorithm 1 Blockwise Ridge with Cyclic Refinement

Input: Features $X \in \mathbb{R}^{n \times d}$, labels $Y \in \mathbb{R}^{n \times c}$
Parameters: K blocks, m features per block, S sweeps
Hyperparameters: λ (ridge), α (damping), feature families
Experimental configuration: $K = 6$, $m = 192$ (1,152 total)
Output: Coefficients $W_1, \ldots, W_K$
Generate random feature maps $\{\phi_k\}_{k=1}^K$
Initialize $W_1, \ldots, W_K$ by the forward residual cascade
for $s = 2$ to S **do**
  for $k = 1$ to K **do**
    $R \leftarrow Y - \sum_{j \neq k} \phi_j(X) W_j$
    $\tilde{W}_k \leftarrow \text{ridge}(\phi_k(X), R)$
    $W_k \leftarrow (1 - \alpha) W_k + \alpha \tilde{W}_k$
  end for
end for
Return: $W_1, \ldots, W_K$

B.3 Feature Standardization

Before solving each block, we standardize features to zero mean and unit variance. The standardization parameters (μ_k, σ_k) are stored and applied during inference. This ensures numerical stability and prevents scale imbalances across blocks. We absorb the inverse standardization into the learned coefficients, so the deployed model operates directly on raw features without requiring a separate standardization step per block.

B.4 Feature Families

We support four random feature families:

ReLU (refinement and block ablations): $\phi(x) = \text{ReLU}(Wx + b)$ where $W \sim \mathcal{N}(0, I)$, $b \sim \text{Unif}(-0.5, 0.5)$

Cosine: $\phi(x) = \cos(Wx + b)$ where $W \sim \mathcal{N}(0, I)$, $b \sim \text{Unif}(0, 2\pi)$ (Random Fourier Features; Rahimi & Recht, 2007)

Tanh: $\phi(x) = \tanh(Wx + b)$ where $W \sim \mathcal{N}(0, I)$, $b \sim \text{Unif}(-0.5, 0.5)$

RBF (pooling and resource comparisons): $\phi(x) = \exp(-\gamma \|x - c_i\|^2)$ where centers c_i are selected via random sampling; the reported pooling batch uses $\gamma = 0.5$

Hybrid mode: Cycles through relu→cos→tanh across blocks to increase diversity, similar to ensemble methods. Evaluated only in the feature family ablation.

The manuscript contains multiple experimental batches rather than one universal feature family: pooling and resource comparisons use standardized RBF center banks, while refinement and block-count ablations use ReLU maps. Within each batch, all compared methods use the same fixed maps and preprocessing. Cosine and tanh are reserved for the feature-family ablation.

B.5 Hyperparameter Selection

Ridge parameter: $\lambda = 0.01$ is optimal across datasets. We tested $\{10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and found 10^{-2} provides best validation performance.

Damping factor: $\alpha = 0.8$ provides best convergence-stability trade-off for multi-sweep mode. We tested $\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ and observed that $\alpha < 0.7$ slows convergence while $\alpha > 0.9$ can cause oscil-

lations. The choice $\alpha = 0.8$ balances numerical stability (avoiding overshooting near the minimum) with convergence speed (maintaining sufficient step size).

Block configuration: Experimental results use $K = 6$ blocks of width $m = 192$ (total capacity 1,152) for consistency with the initial setup. In the seeded ablation in Section 5.4, $K = 1$ gives the highest accuracy, while $K = 6$ provides a favorable speed–accuracy trade-off. These observations are empirical rather than universal defaults across feature widths or datasets.

C Extended Experimental Results

C.1 Convergence Analysis Details

Figure 3(b) presents the CIFAR-100-ViT convergence trend in the main text. Table 5 provides a separate CIFAR-10-ResNet18 convergence analysis across block counts and sweep counts.

Table 5: Test accuracy gap to joint ridge solution (%) on CIFAR-10-ResNet18. Empirical convergence across sweep counts.

Blocks	1-sweep	3-sweep	5-sweep	10-sweep	20-sweep	Joint Ridge
4	0.46	0.26	0.19	0.14	0.09	84.88
6	0.72	0.51	0.42	0.33	0.21	84.88
8	1.06	0.68	0.53	0.27	0.13	84.88

Key observations: (1) The measured accuracy gap decreases monotonically in these runs, (2) more blocks have larger initial gaps and slower convergence, (3) 20 sweeps take 5–7s in this experiment, and (4) the gains diminish after approximately 10–15 sweeps.

C.2 Memory-Speed Trade-off Analysis

Figure 4 is discussed in the main text. Our method stores the full dataset on GPU for analytic QR solving, whereas backpropagation uses mini-batch DataLoaders that process 1024 samples at a time.

Interpretation: The cached-feature setting trades higher accelerator memory for shorter fitting time relative to the mini-batch baselines in this experiment. The absolute trade-off depends on dataset size, block configuration, and hardware.

When unfavorable: (1) Memory-constrained environments (<2 GB GPU), (2) Datasets exceeding GPU memory (streaming variants possible but slower), (3) Shared GPU resources where memory allocation matters.

C.3 Block Configuration Trade-offs

Figure 5 analyzes how block configuration affects accuracy, fitting time, and estimated model memory under fixed total capacity (1,152 features) on CIFAR-100-ViT.

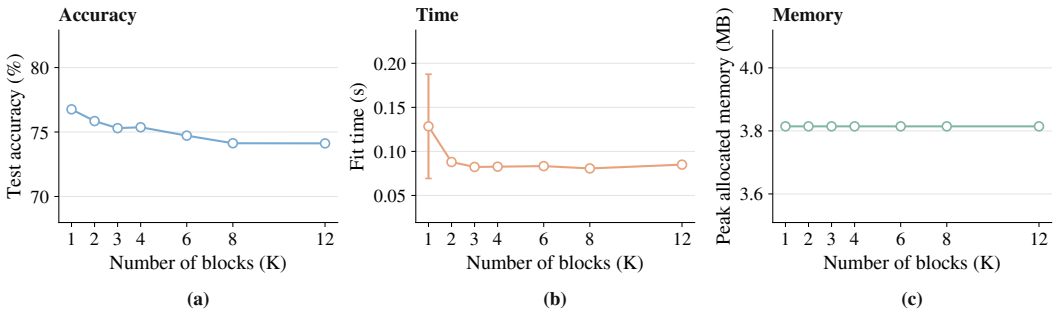


Figure 5: Block-configuration trade-offs on CIFAR-100-ViT under fixed total capacity of 1,152 features. Panels (a)–(c) report test accuracy, fitting time, and estimated model memory as the number of blocks varies; values are means over three independently seeded feature banks, with error bars denoting sample standard deviations.

C.4 Numerical Stability Validation

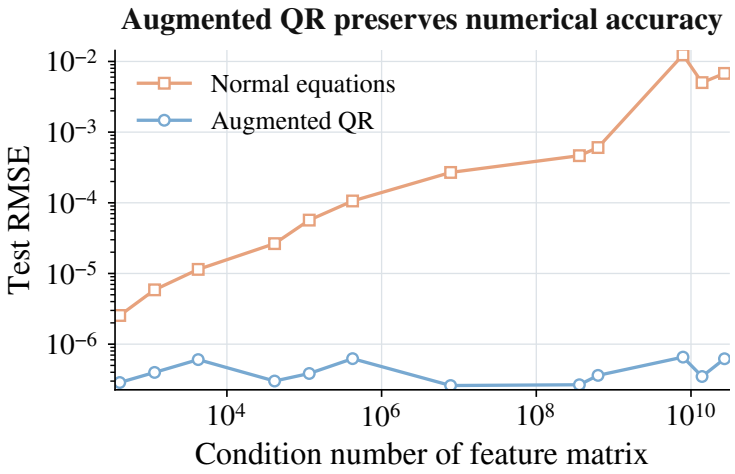


Figure 6: Numerical stability of ridge solvers under increasing feature-matrix condition numbers. Normal equations exhibit rapidly increasing test RMSE, whereas augmented QR maintains substantially lower error across the tested range.

Figure 6 provides the stability evidence: as the feature-matrix condition number increases, the test RMSE of normal equations grows sharply, while augmented QR remains comparatively stable. The plotted values are single deterministic runs over the tested synthetic matrices; no unsupported error bars are reported. Training time is similar in this diagnostic, while the numerical accuracy of augmented QR is substantially better under ill-conditioning.

C.5 Additional Dataset Results

Table 6 provides per-dataset breakdown of performance metrics.

Table 6: Per-dataset performance breakdown: accuracy, training time, peak memory, and speedup vs joint ridge.

Dataset	Accuracy (%)	Time (s)	Memory (MB)	Speedup
MNIST-HOG	97.41 ± 0.06	0.203 ± 0.002	760.6	1.13×
CIFAR-10-ResNet18	82.61 ± 0.16	0.154 ± 0.001	402.3	1.13×
CIFAR-100-ResNet18	63.35 ± 0.12	0.156 ± 0.002	433.5	1.14×
CIFAR-100-ViT	76.74 ± 0.17	0.164 ± 0.001	479.7	1.11×
flowers102-DINOv2	99.56 ± 0.02	0.017 ± 0.001	28.2	1.57×

D Experimental Configuration

All experiments use the following configuration unless stated otherwise:

D.1 Hardware and Software

Table 7: Hardware and software configuration.

Component	Configuration
GPU	NVIDIA RTX 3090 (24 GB VRAM)
CUDA	11.8
PyTorch	2.0
Python	3.10
Operating system	Ubuntu 20.04 LTS

D.2 HYPERPARAMETERS

Table 8: Hyperparameter configuration used in the experiments.

Parameter	Value	Scope
Ridge coefficient λ	0.01	All experiments
Number of blocks K	6	Convergence and 1,152-feature ablations
Block width m	192	$K = 6$ configuration
Total width / block width	$2,048$ / 256	Multi-dataset benchmark
Feature family	RBF (pooling/resource); ReLU (refinement/ablations)	By experiment batch
Damping factor α	0.8	Multi-sweep refinement

D.3 BASELINE CONFIGURATION

Table 9: Baseline model and optimizer configurations.

Baseline	Configuration
Logistic Probe	`LogisticRegression(C=1.0, max_iter=1000, solver=lbfgs)`
MLP	Two hidden layers [512 $\rightarrow$ 256 $\rightarrow$ num_classes]; batch size 1024; 50 epochs
SGD	Learning rate 0.01; momentum 0.9
Adam	Learning rate 0.001; betas $(0.9, 0.999)$
AdamW	Learning rate 0.001; weight decay 0.01
RMSprop	Learning rate 0.001; $\alpha = 0.99$
ELM	1,152 hidden units; ReLU activation; $\lambda = 0.01$

D.4 TIMING METHODOLOGY

Table 10: Timing protocol used for reported fitting-time measurements.

Item	Protocol
Included	Classifier fitting time only; features are pre-cached on the GPU
Excluded	Feature extraction time, treated as a one-time cost amortized across runs
Synchronization	`torch.cuda.synchronize()` before and after each timed region
Averaging	Mean over 3 runs with different random seeds
Warm-up	One warm-up run before timing to avoid CUDA initialization overhead
Reproducibility	Fixed random seeds, data order, random feature maps, and numerical backend

D.5 DATASET SPECIFICATIONS

Table 11: Dataset specifications (features pre-extracted and cached on disk as PyTorch tensors).

Dataset	Feature Extractor	Feature Dim	Train Samples	Test Samples
MNIST-HOG	Hand-crafted HOG	1,296	60,000	10,000
CIFAR-10-ResNet18	ResNet-18 (frozen)	512	50,000	10,000
CIFAR-100-ResNet18	ResNet-18 (frozen)	512	50,000	10,000
CIFAR-100-ViT	ViT-B/16 (frozen)	768	50,000	10,000
flowers102-DINOv2	DINOv2-small (frozen)	384	1,020	6,149

E DISCUSSION

E.1 WHEN TO USE THIS METHOD

Our method is most useful when: (1) features are frozen and pre-cached from pre-trained encoders, (2) classifiers must be retrained frequently, and (3) avoiding iterative optimization is valuable. The blockwise implementation can reduce peak working memory relative to a matched joint solve, but still requires the cached feature matrix to fit on the accelerator for the reported timings.

Our method is less suitable when: (1) features must be learned jointly with the classifier, (2) features are not pre-cached, (3) memory is severely constrained, or (4) absolute accuracy is paramount and a one-pass approximation gap is unacceptable.

E.2 CONFIGURATION GUIDELINES

Based on the ablations in Section 5.4, the one-block configuration gives the highest observed accuracy, while six blocks provide a favorable speed–accuracy trade-off for the evaluated CIFAR-100-ViT setting with 1,152 total features. Cyclic refinement is appropriate when accuracy is more important than fitting time: in the corresponding six-block experiment, additional sweeps substantially reduce the gap to the joint solution, at the cost of extra solves.

E.3 LIMITATIONS

The method requires the cached training features to reside on the accelerator for the reported analytic timings, which can result in higher memory use than mini-batch training. A single sweep can lose accuracy relative to tuned backpropagation, and the classifier head remains limited by the frozen representation and its linear output layer. The experiments focus on medium-scale frozen-feature benchmarks; out-of-core implementations and substantially larger datasets remain future work.