SoftServe 🍦: A Scalable Quasi-Newton Method for Deep Learning

Anonymous authors
Paper under double-blind review

Abstract

Quasi-Newton (QN) methods have long been among the most effective methods for large-scale unconstrained convex optimization. Two obstacles have limited their use in deep learning: non-convexity and enormous parameter sizes. We introduce SoftServe, a family of QN methods designed to overcome these obstacles without line searches or ad hoc curvature corrections. SoftServe derives positive-definite curvature estimates from the variational objective of Berglund et al. (2025), even in the presence of negative curvature. We develop diagonal and Kronecker-factored variants that preserve positive definiteness by construction and scale to massive neural networks. Finally, SoftServe relies on the stable coupled Newton–Schulz iteration for the required matrix operations, replacing costly matrix decompositions with GPU-friendly matrix multiplications. SoftServe excels on problems that are severely ill-conditioned, including tasks such as recurrent networks, deep autoencoders, physics-informed neural networks, and a 136M-parameter physics-informed diffusion model, often achieving lower losses than established baselines including Adam, Muon, and SOAP.

1 Introduction

Quasi-Newton methods, such as the celebrated BFGS method (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), are among the most effective methods for large-scale, convex deterministic optimization. Deep learning has led to renewed interest in adapting these methods to train large networks (Goldfarb et al., 2020; Ren et al., 2022). However, applying these methods to deep learning requires curvature estimates that can accommodate non-convex objectives and gradient noise while remaining computationally efficient.

To see how curvature enters the optimization update, consider a generic second-order method. For parameters $x \in \mathbb{R}^D$, second-order methods typically take steps of the form

$$x^+ = x - \gamma \boldsymbol{H} g.$$

Here x^+ is the updated parameter vector, and $\gamma > 0$ is the step size. The vector $g \in \mathbb{R}^D$ is the (stochastic) gradient at x, and the matrix $\boldsymbol{H} \in \mathbb{R}^{D \times D}$ is an estimate of the inverse Hessian.

Non-convexity of the objective poses a fundamental challenge for second-order optimization methods because the true Hessian could have negative eigenvalues. If the estimate $\boldsymbol{H}$ has a negative eigenvalue, then the update direction $-\boldsymbol{H}g$ could be an ascent direction, which would increase the loss function. This is one of the reasons why first-order methods are often preferred in deep learning: one can always guarantee descent (in expectation). For a second-order method to be applicable in deep learning, the estimate $\boldsymbol{H}$ must always be positive definite, even in the presence of directions of negative curvature.

For quasi-Newton methods, such as BFGS, maintaining positive definiteness is at odds with the main tool used to measure curvature, namely the *secant equation*. In the presence of negative curvature, enforcing the secant equation can inadvertently force the Hessian estimate to become indefinite, as we will show in Section 2. The standard remedy is to use sophisticated line search procedures to only move in directions of positive curvature (Nocedal

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.

& Wright, 2006; Wolfe, 1969). But these line searches require access to the full batch loss, which is too costly and not feasible in the stochastic setting of deep learning.

Beyond the challenges arising from non-convexity and stochastic objectives, the sheer scale of modern deep learning models makes storing and updating dense curvature estimates prohibitively expensive. A standard approach for reducing this cost is the limited memory approach of L-BFGS (Liu & Nocedal, 1989), which typically uses only 10 pairs of parameters and gradients to implicitly build the estimate $\boldsymbol{H}$. But storing 10 pairs of parameters and gradients would require storing 10 copies of our model and 10 copies of the gradient, increasing the memory storage by 20-fold. This is infeasible for all but small models.

To scale to modern deep learning, it is common to approximate the matrix $\boldsymbol{H}$ by a diagonal matrix (Duchi et al., 2011; Tieleman & Hinton, 2012; Kingma & Ba, 2015) or a Kronecker-structured matrix (Martens & Grosse, 2015; Gupta et al., 2018). For quasi-Newton optimization, while Kronecker-structured updates have been proposed for applying BFGS to deep learning (Goldfarb et al., 2020; Ren et al., 2022), this approach still needs to be applied with mechanisms for handling non-convexity and stochasticity, limiting its applicability.

A promising new direction is to reconsider how curvature enters the quasi-Newton update. Recently, Berglund et al. (2025) show that by replacing the secant equation with a soft secant penalty we can ensure that our Hessian estimate is positive definite, even in the presence of negative curvature. But Berglund et al. (2025) only consider dense estimates, making their approach infeasible for deep learning.

We introduce SoftServe (Soft **S**ecant **E**quation **R**egularized with **V**ariational **E**ntropy), a quasi-Newton framework that combines the flexibility of a soft secant constraint (Li, 2019; Berglund et al., 2025) with the computational advantages of a structured matrix approximation of the inverse Hessian. Optimizing within diagonal and Kronecker-structured families yields novel updates that preserve positive definiteness, without full batch evaluations or line searches. We develop efficient algorithms for computing these updates using modern GPU-friendly numerical linear algebra methods, such as variants of Newton-Schulz (Amsel et al., 2026; Higham, 2008), allowing SoftServe to scale to large neural networks.

We then demonstrate the utility of SoftServe on several deep learning tasks where curvature information is fundamental—such as severely ill-conditioned tasks—including recurrent networks, autoencoders, and physics-informed models (such as a 136M-parameter diffusion model). On these tasks we find that SoftServe often outperforms modern methods such as Adam, Muon, and SOAP. A Python implementation of SoftServe is available at `https://anonymous.4open.science/r/softserve-anonymous-602B`.

2 Quasi-Newton Preliminaries

Here we introduce our notation and review important quasi-Newton (QN) ideas that are relevant to SoftServe. For simplicity, in this section we represent all parameters as a flattened vector $x \in \mathbb{R}^D$. We introduce the setting of deep learning in subsequent sections, where parameters can be matrices.

Notation. We write $\mathbb{S}^d_{++}$ and $\mathbb{S}^d_{+}$ for the symmetric positive definite and positive semidefinite $d \times d$ matrices, $\boldsymbol{I}_d$ for the identity, and we define $\|v\|^2_{\boldsymbol{M}} := v^\top \boldsymbol{M} v$ for $\boldsymbol{M} \in \mathbb{S}^d_{++}$.

2.1 The BFGS algorithm

Let f be a continuous, twice differentiable function on $\mathbb{R}^D$. Given iterates x_t and gradient evaluations $\nabla f(x_t)$, quasi-Newton methods use the following iterate and gradient differences

$$\begin{aligned} s_t &:= x_{t+1} - x_t, \\ y_t &:= \nabla f(x_{t+1}) - \nabla f(x_t), \end{aligned}$$

to estimate curvature. Indeed, using Taylor's theorem we have that

$$y_t = \boldsymbol{B}_t s_t, \tag{1}$$

where $\boldsymbol{B}_t := \int_0^1 \nabla^2 f(x_t + \tau s_t)\, d\tau$ is the averaged Hessian. Quasi-Newton methods use Eq. 1 to find an estimate of the inverse Hessian $\boldsymbol{H}$ by enforcing the *secant equation*

$$\boldsymbol{H} y_t = s_t. \tag{2}$$

However, the secant equation is underdetermined: $\boldsymbol{H}$ has D^2 unknown entries, while Eq. 2 imposes only D constraints. Thus, to obtain a unique update, one must impose an additional regularization criterion. Quasi-Newton methods resolve this by choosing $\boldsymbol{H}$ that both satisfies the secant equation and is close to the previous estimate $\boldsymbol{H}_t$ in some sense.

For instance, the BFGS method chooses the matrix $\boldsymbol{H}$ that satisfies the secant equation and is closest to $\boldsymbol{H}_t$ in a weighted Frobenius norm (Nocedal & Wright, 2006); see Appendix A.1. We adopt an equivalent characterization of the BFGS update due to Fletcher (1991), where closeness is measured by the *log-determinant divergence*. This formulation will be the starting point for SoftServe. For $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{S}^d_{++}$ the log-determinant divergence is defined as

$$\mathscr{D}(\boldsymbol{P}; \boldsymbol{Q}) := \operatorname{tr}(\boldsymbol{P}\boldsymbol{Q}^{-1}) - \log\det(\boldsymbol{P}\boldsymbol{Q}^{-1}) - d. \tag{3}$$

This divergence is strictly convex in $\boldsymbol{P}$ (given fixed $\boldsymbol{Q}$) and also is equivalent (up to a constant) to the Kullback-Leibler divergence between two Gaussians with the same mean and covariances $\boldsymbol{P}$ and $\boldsymbol{Q}$. Fletcher (1991) showed that, when $s_t^\top y_t > 0$, the BFGS update is given by the solution of the following optimization problem:

$$\boldsymbol{H}_{t+1}^{\text{BFGS}} = \underset{\boldsymbol{H} \in \mathbb{S}^D_{++}}{\arg\min}\ \mathscr{D}(\boldsymbol{H}_t; \boldsymbol{H}) \quad \text{subject to} \quad \boldsymbol{H} y_t = s_t. \tag{4}$$

What happens in the non-convex and stochastic setting? The requirement $s_t^\top y_t > 0$ is called the *curvature condition* and it always holds for convex objectives, but can fail for nonconvex objectives. Indeed, if $s_t^\top y_t < 0$, then any matrix $\boldsymbol{H}$ satisfying the secant equation must also satisfy $s_t^\top y_t = y_t^\top \boldsymbol{H} y_t < 0$, and thus cannot be positive definite.

In stochastic settings, the construction of the gradient difference y_t requires additional care. Let $\nabla f_\xi(x)$ denote a stochastic gradient evaluated with random draw ξ, which specifies the minibatch and other sources of randomness such as data augmentation or dropout masks. In this case, we compute $y_t = \nabla f_\xi(x_{t+1}) - \nabla f_\xi(x_t)$ using the same draw at both endpoints, in order to ensure that the gradient difference reflects the curvature for a single sampled loss. Enforcing the secant equation Eq. 2 in the stochastic setting is also problematic, since Eq. 2 would force $\boldsymbol{H}$ to exactly fit the curvature of just one stochastic loss.

2.2 Soft quasi-Newton

Soft QN (quasi-Newton) builds on the alternative variational formulation of BFGS given in Eq. 4, relaxing the secant constraint by transforming it into a least-squares penalty in the objective function (Berglund et al., 2025). Given $\lambda > 0$ and the current estimate $\boldsymbol{H}_t$, the Soft QN algorithm approximates the inverse Hessian by solving

$$\boldsymbol{H}_{t+1} = \underset{\boldsymbol{H} \in \mathbb{S}^D_{++}}{\arg\min}\ \|s_t - \boldsymbol{H} y_t\|^2_{\boldsymbol{H}^{-1}} + \tfrac{1}{\lambda}\mathscr{D}(\boldsymbol{H}_t; \boldsymbol{H}). \tag{5}$$

The first term in Eq. 5 penalizes the secant residual; the second term measures how far $\boldsymbol{H}_t$ is from $\boldsymbol{H}$ in log-determinant divergence Eq. 3. The parameter $\lambda > 0$ controls the tradeoff between the two terms.

Eq. 5 has several nice properties. First, it is scale invariant and strictly convex in $\boldsymbol{H}^{-1}$. Second, there is a closed-form solution for $\boldsymbol{H}_{t+1}$, given in Eq. A.4. Third, the BFGS update is also a solution to Eq. 5 in the limit as $\lambda \to \infty$ and $s_t^\top y_t > 0$ (see Berglund et al. (2025) and Appendix A.2). Finally and crucially, Soft QN remains well-defined and positive definite for every finite pair, including those with negative curvature $s_t^\top y_t \le 0$.

However, computing the solution of Eq. 5 costs $\mathcal{O}(D^2)$ operations, which is computationally expensive, and the memory cost of storing the D^2 matrix $\boldsymbol{H}$ is prohibitive in deep learning. We circumvent both the operational and memory costs by restricting to structured matrices.

3 SoftServe: Quasi-Newton Updates over Structured Families

To scale to deep neural networks, we consider two choices for structured families $\mathcal{H} \subset \mathbb{S}^D_{++}$: a diagonal matrix family and Kronecker-structured matrix family. SoftServe then solves Eq. 5 for an inverse Hessian approximation $\boldsymbol{H}$ over the structured family $\mathcal{H}$, instead of over dense matrices in $\mathbb{S}^D_{++}$. This optimization yields novel and efficiently-computable updates, which we call SoftServe-Diag and SoftServe-Kron.

Instead of the $\mathcal{O}(D^2)$ cost of Soft QN, SoftServe-Diag reduces the memory and computational cost to $\mathcal{O}(D)$, which is comparable to that of Adam (Kingma & Ba, 2015). SoftServe-Kron reduces the costs to $\mathcal{O}(n^2 + m^2)$ for a $D = n \times m$ parameter, which is the cost of Shampoo (Gupta et al., 2018) and KFAC (Martens & Grosse, 2015).

3.1 Diagonal approximation

SoftServe-Diag restricts the inverse Hessian estimate $\boldsymbol{H}$ to be diagonal. Thus we need only store one positive scale per coordinate, $\boldsymbol{H} = \operatorname{diag}(h)$ with $h \in \mathbb{R}^D_{++}$. Both terms of the objective in Eq. 5 then split into sums over coordinates $h_{t+1,i}$. Each coordinate is a scalar problem with a unique, closed-form solution $h_{t+1,i} > 0$ (Lemma B.1), given by

$$h_{t+1,i} = \frac{2\left(h_{t,i} + \lambda s_{t,i}^2\right)}{1 + \sqrt{1 + 4\lambda y_{t,i}^2\left(h_{t,i} + \lambda s_{t,i}^2\right)}}, \qquad i = 1, \ldots, D. \tag{6}$$

Thus, the matrix estimate is $\boldsymbol{H}_{t+1} = \operatorname{diag}(h_{t+1})$. Though SoftServe-Diag is inexpensive, and can perform reasonably well, it ignores all interactions between coordinates.

3.2 Kronecker-structured approximation

Neural network parameters are often naturally represented as matrices. Let $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ denote a layer's weight matrix and $\boldsymbol{G} := \nabla_{\boldsymbol{X}} f \in \mathbb{R}^{n \times m}$ denote its gradient. Thus the Hessian matrix associated with $\boldsymbol{X}$ has $n^2 m^2$ entries. We can reduce this storage cost by imposing a Kronecker structure:

$$\boldsymbol{H} = \boldsymbol{L} \otimes \boldsymbol{R}, \quad \text{where} \quad \boldsymbol{L} \in \mathbb{S}^m_{++}, \quad \boldsymbol{R} \in \mathbb{S}^n_{++}. \tag{7}$$

Here $\otimes$ denotes the *Kronecker product*: if $\boldsymbol{L} = [a_{ij}]$, then $\boldsymbol{L} \otimes \boldsymbol{R}$ is the $nm \times nm$ block matrix whose $(i,j)^{\text{th}}$ block is $a_{ij}\boldsymbol{R}$. Storing the two factors $\boldsymbol{L}$ and $\boldsymbol{R}$ requires only $m^2 + n^2$ entries. Applying $\boldsymbol{H}$ to the gradient also requires only two matrix products: $\boldsymbol{H}g = \operatorname{vec}(\boldsymbol{RGL})$, where $g = \operatorname{vec}(\boldsymbol{G})$ and $\operatorname{vec}(\cdot)$ stacks columns.

Unfortunately there is no closed-form solution when minimizing the soft quasi-Newton objective Eq. 5 over both $\boldsymbol{L}$ and $\boldsymbol{R}$. But there is a unique solution if we solve for only one factor at a time. We therefore update them alternately, holding one fixed while optimizing the other. As we now show, each conditional minimization reduces to solving a quadratic matrix equation with a unique positive-definite solution (see Lemma B.3).

To see this, we first reshape the secant vectors into matrices $\boldsymbol{S}_t, \boldsymbol{Y}_t \in \mathbb{R}^{n \times m}$, with $\operatorname{vec}(\boldsymbol{S}_t) = s_t$ and $\operatorname{vec}(\boldsymbol{Y}_t) = y_t$. Holding $\boldsymbol{R} = \boldsymbol{R}_t$ fixed and minimizing the terms in Eq. 5 involving $\boldsymbol{L}$, we obtain the conditional update

$$\boldsymbol{L}_{t+1} = \operatorname*{arg\,min}_{\boldsymbol{L} \in \mathbb{S}^m_{++}} \quad \operatorname{tr}(\boldsymbol{U}_L \boldsymbol{L}) + \operatorname{tr}(\boldsymbol{V}_L \boldsymbol{L}^{-1}) + \log\det \boldsymbol{L}, \tag{8}$$

where we define the auxiliary matrices:

$$\boldsymbol{U}_L := \tfrac{\lambda}{n} \boldsymbol{Y}_t^\top \boldsymbol{R}_t \boldsymbol{Y}_t \quad \text{and} \quad \boldsymbol{V}_L := \boldsymbol{L}_t + \tfrac{\lambda}{n} \boldsymbol{S}_t^\top \boldsymbol{R}_t^{-1} \boldsymbol{S}_t. \tag{9}$$

Differentiating Eq. 8 with respect to $\boldsymbol{L}$ and setting the derivative to zero, yields the quadratic matrix equation

$$\boldsymbol{L}\,\boldsymbol{U}_L\,\boldsymbol{L} + \boldsymbol{L} = \boldsymbol{V}_L. \tag{10}$$

Here, because $\boldsymbol{U}_L \succeq 0$ and $\boldsymbol{V}_L \succ 0$, this equation has a unique, positive-definite solution $\boldsymbol{L}_{t+1}$, which we provide in Eq. 13 and discuss further in Section 4.1.

We then update the factor $\boldsymbol{R}$ by minimizing Eq. 5 in $\boldsymbol{R}$ and holding $\boldsymbol{L}_{t+1}$ fixed:

$$\boldsymbol{R}_{t+1} = \underset{\boldsymbol{R}\in\mathbb{S}^n_{++}}{\arg\min} \quad \operatorname{tr}(\boldsymbol{U}_R\boldsymbol{R}) + \operatorname{tr}(\boldsymbol{V}_R\boldsymbol{R}^{-1}) + \log\det\boldsymbol{R}, \tag{11}$$

where the auxiliary matrices are now defined as

$$\boldsymbol{U}_R := \tfrac{\lambda}{m}\boldsymbol{Y}_t\boldsymbol{L}_{t+1}\boldsymbol{Y}_t^\top \quad \text{and} \quad \boldsymbol{V}_R := m^{-1}[\operatorname{tr}(\boldsymbol{L}_t\boldsymbol{L}_{t+1}^{-1})\boldsymbol{R}_t + \lambda\boldsymbol{S}_t\boldsymbol{L}_{t+1}^{-1}\boldsymbol{S}_t^\top]. \tag{12}$$

Solving Eq. 11 for $\boldsymbol{R}_{t+1}$ reduces to solving an analogous quadratic matrix equation to Eq. 10, $\boldsymbol{R}\,\boldsymbol{U}_R\,\boldsymbol{R} + \boldsymbol{R} = \boldsymbol{V}_R$, which again has a unique, positive-definite solution (Lemma B.3).

The alternating updates replace computations on the full inverse-Hessian estimate $\boldsymbol{H}$ with operations on the two smaller factors $\boldsymbol{L}$ and $\boldsymbol{R}$. Nevertheless, directly solving the quadratic matrix equations can be expensive for wide layers, with cost $\mathcal{O}(n^3 + m^3 + n^2m + nm^2)$. Instead of using a direct solver, we will use GPU-friendly Newton-Schulz approximations, as discussed in the next section. Using Newton-Schulz approximations was also central to Muon's effectiveness at scale (Amsel et al., 2026; Liu et al., 2025).

4 Practical implementation of SoftServe for deep learning

Here we discuss strategies to make SoftServe practical for training deep neural networks.

4.1 Efficient Kronecker updates

SoftServe-Kron requires solving quadratic matrix equations (Cai et al., 2024), which are particular instances of the algebraic Riccati equation (Lancaster & Rodman, 1995). Here we discuss how to approximately solve this matrix equation using a GPU friendly approach. These approximations are key for scaling SoftServe up to large neural networks.

Both Kronecker factor updates in Eq. 8 and Eq. 11 reduce to solving a quadratic matrix equation of the form $\boldsymbol{Q}\boldsymbol{U}\boldsymbol{Q} + \boldsymbol{Q} = \boldsymbol{V}$ with $\boldsymbol{U} \succeq 0$ and $\boldsymbol{V} \succ 0$. Its unique positive-definite solution can be expressed using principal matrix square roots $\boldsymbol{V}^{1/2}$ (see Appendix B.3):

$$\boldsymbol{Q} = 2\boldsymbol{V}^{1/2}\left[\boldsymbol{I}_d + \left(\boldsymbol{I}_d + 4\boldsymbol{V}^{1/2}\boldsymbol{U}\boldsymbol{V}^{1/2}\right)^{1/2}\right]^{-1}\boldsymbol{V}^{1/2}. \tag{13}$$

Direct evaluation of Eq. 13 requires two matrix square roots and a matrix inverse. For Kronecker updates, the coefficients admit a natural Gram matrix factorization $\boldsymbol{U} = \boldsymbol{F}\boldsymbol{F}^\top$, given in Eq. B.22. Our implementation exploits this structure through an equivalent Gram formulation of Eq. 13, which evaluates a square root of $\boldsymbol{I} + 4\boldsymbol{F}^\top\boldsymbol{V}\boldsymbol{F}$ and an inverse, which avoids the separate square roots of $\boldsymbol{V}$. See Appendix B.3 and C.1 for details.

While the matrix functions in Eq. 13 can be computed using, for instance singular value decomposition (SVD), most methods for computing the SVD are not well suited for GPUs (Heavner et al., 2023). To compute the matrix square root efficiently on GPUs, we use the coupled Newton-Schulz iteration (Higham, 1997; 2008), which is based only on matrix multiplications and additions. For a positive definite matrix $\boldsymbol{M}$, coupled Newton-Schulz jointly approximates its square root $\boldsymbol{M}^{\frac{1}{2}}$ and its inverse square root $\boldsymbol{M}^{-\frac{1}{2}}$. Matrix inverses are likewise approximated using Newton-Schulz iterations. See Appendix C.2 for details.

Algorithm 1 SoftServe-Kron: high-level training iteration (details in Algorithm D.1)

Require: Parameters $\boldsymbol{X}_t$; factors $\boldsymbol{L}_t, \boldsymbol{R}_t$; momentum estimate $\hat{\boldsymbol{M}}_t$; step size γ_t; penalty λ.

1: Take the normalized preconditioned step in Eq. 14, using

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \gamma_t \frac{\boldsymbol{R}_t \hat{\boldsymbol{M}}_t \boldsymbol{L}_t}{\sqrt{\operatorname{tr}\left(\hat{\boldsymbol{M}}_t^\top \boldsymbol{R}_t \hat{\boldsymbol{M}}_t \boldsymbol{L}_t\right)}}$$

2: Evaluate stochastic gradient, difference of gradients $\boldsymbol{Y}_t$ and difference of iterates $\boldsymbol{S}_t$

3: Update Kronecker factors by solving quadratic matrix equations (via Eq. 13 or Eq. B.22):

$$\text{Update } \boldsymbol{L}_{t+1} \text{ by solving } \boldsymbol{L}\boldsymbol{U}_L\boldsymbol{L} + \boldsymbol{L} = \boldsymbol{V}_L$$
$$\text{Update } \boldsymbol{R}_{t+1} \text{ by solving } \boldsymbol{R}\boldsymbol{U}_R\boldsymbol{R} + \boldsymbol{R} = \boldsymbol{V}_R,$$

where the matrices $\boldsymbol{U}_L, \boldsymbol{V}_L$ are given in Eq. 9, and $\boldsymbol{U}_R, \boldsymbol{V}_R$ are given in Eq. 12.

4: **return** $\boldsymbol{X}_{t+1}$

4.2 Training with SoftServe

To make SoftServe practical, we make use of several now-standard practices in deep learning. The full procedure includes momentum updates, Nesterov extrapolation or bias correction (Kingma & Ba, 2015), details of which are in Appendix D.1.

Algorithm 1 summarizes one training iteration of SoftServe-Kron applied to one matrix parameter $\boldsymbol{X} \in \mathbb{R}^{n\times m}$. SoftServe-Diag follows the same procedure replacing the Kronecker updates with the diagonal update in Eq. 6, and detailed in Algorithm D.2

One of main changes we have introduced in Algorithm 1 is that we use a *normalized* update, see line 1. In vectorized notation, this normalized update is given by

$$x_{t+1} = x_t - \gamma_t \boldsymbol{H}_t \hat{m}_t \Big/ \sqrt{\hat{m}_t^\top \boldsymbol{H}_t \hat{m}_t}, \tag{14}$$

where $x_t = \operatorname{vec}(\boldsymbol{X}_t), \hat{m}_t = \operatorname{vec}(\hat{\boldsymbol{M}}_t)$ and $\boldsymbol{H}_t = \boldsymbol{L}_t \otimes \boldsymbol{R}_t$. Normalized updates are often more effective in deep learning (Jordan et al., 2024; Pethick et al., 2025; Crawshaw et al., 2026), which we also confirmed in our initial experiments.

Finally, to amortize the cost of refreshing the curvature estimate, we hold $\boldsymbol{H}_t$ fixed for K training steps. At the beginning of each window, we save the parameters, stochastic gradient, and random draw. At the end, we reevaluate the gradient using the saved draw, forming a secant pair from the endpoint differences, see Appendix D.1 for details.

5 Related Work

Stochastic and structured Quasi-Newton. Classical QN methods enforce the secant equation while changing the metric as little as possible (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970; Dennis & Moré, 1977; Nocedal & Wright, 2006). Stochastic variants keep a dense or limited-memory metric (Liu & Nocedal, 1989) and protect it from bad pairs by: using damping (Schraudolph et al., 2007), curvature pairs from Hessian-vector products on large batches (Byrd et al., 2016), damped updates (Wang et al., 2017), overlapping batches (Berahas et al., 2016), diagonal rescaling (Bordes et al., 2009), or a secant penalty (Irwin & Haber, 2023). K-BFGS (Goldfarb et al., 2020) uses a per-layer Kronecker factorization of BFGS, and relies on damping for a resulting positive definite curvature estimate.

Matrix-aware methods. There are now a growing number of methods which are specialized to the matrix parameters in linear layers. K-FAC (Martens & Grosse, 2015), Shampoo (Gupta et al., 2018), and SOAP (Vyas et al., 2025) all make use of Kronecker factorizations to form a preconditioner. Muon (Jordan et al., 2024; Liu et al., 2025) is a steepest descent method in the matrix spectral norm.

Related variational objectives. With the mean held fixed, the covariance subproblem in batch-and-match variational inference (Cai et al., 2024) has the same form as the soft

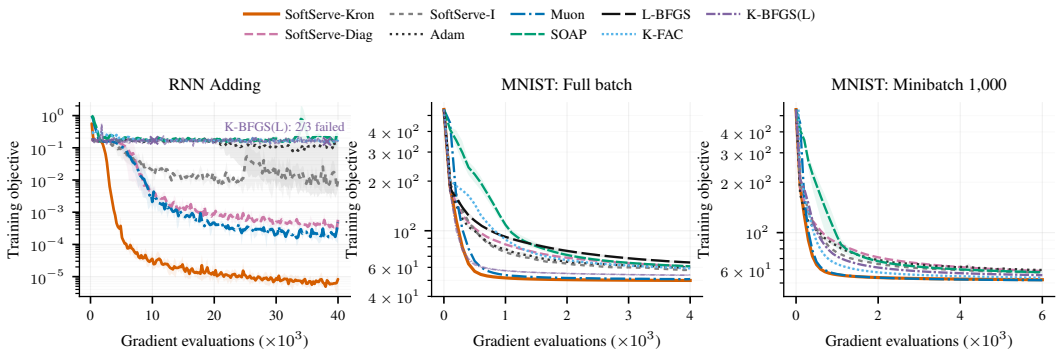


Figure 1: Training objectives against gradient evaluations: prediction MSE for RNN Adding (left), and reconstruction binary cross-entropy with ℓ_2 regularization for full-batch MNIST (middle) and minibatch MNIST (right). Lines show three-seed means; shading spans the seed minimum and maximum. K-BFGS(L) with failed seeds are shown as individual traces.

QN objective in (5), which is where the closed form in Eq. 13 originates (see Section A.3 for details). The least squares secant penalty within the soft QN objective (5) has also been used to design preconditioned variants of SGD called PSGD in (Li, 2018; Pooladzandi & Li, 2024). PSGD then uses a form of multiplicative SGD applied to the secant penalty to estimate the preconditioner.

6 Experiments

We evaluate SoftServe on recurrent sequence learning, image reconstruction, and physics-informed learning, from a small recurrent network to a 136M-parameter diffusion model. We test whether the structured inverse Hessian estimates improve training over first-order and other preconditioned methods. We compare training losses at equal gradient-evaluation budgets, and report prediction errors and wall-clock comparisons in Appendix E.

Comparison protocol. Our baselines are Adam, Muon, SOAP, K-FAC, and K-BFGS(L), with L-BFGS also included for deterministic training. To assess the contribution of the learned inverse Hessian, we include SoftServe-I: the normalized momentum method Eq. 14 obtained by fixing $\boldsymbol{H} = \boldsymbol{I}$. We tune each method's learning rate separately and evaluate the selected configurations on three fresh seeds. All SoftServe runs refresh the inverse Hessian estimate every $K = 10$ updates. Gradient counts include the additional evaluations used for secant pairs, line searches, and loss balancing. In stochastic training, SoftServe uses one extra gradient evaluation every K updates to form a secant pair on the same batch, see detailed pseudo-code in Algorithm D.1. Task-specific settings and learning-rate sweeps are given in Appendix E.

6.1 RNN Adding

The Adding problem tests a recurrent network's ability to retain information over a long sequence. Each input contains 100 numbers sampled uniformly from $[0, 1]$ and markers identifying two positions. The network reads the full sequence and predicts the sum of the two marked numbers. We train a tanh RNN with 16 hidden units by minimizing mean squared error (MSE) on batches of 128 newly sampled sequences. Although the network is small, repeated applications of the recurrent map can cause gradients to vanish or explode (Pascanu et al., 2013). The full loss Hessian in this example is numerically nearly singular at initialization across four seeds (Appendix E.2). This makes Adding an interesting ill-conditioned small-scale test. Indeed, in Figure 1 we see that preconditioning was necessary to reach a reasonable loss, with SoftServe-Kron reaching by far the lowest training loss (Figure 1, left), outperforming our baselines on all three seeds. SoftServe-Kron mean test MSE is 6.6×10^{-6}, compared with 1.5×10^{-4} for Muon and 1.0×10^{-2} for SoftServe-I.

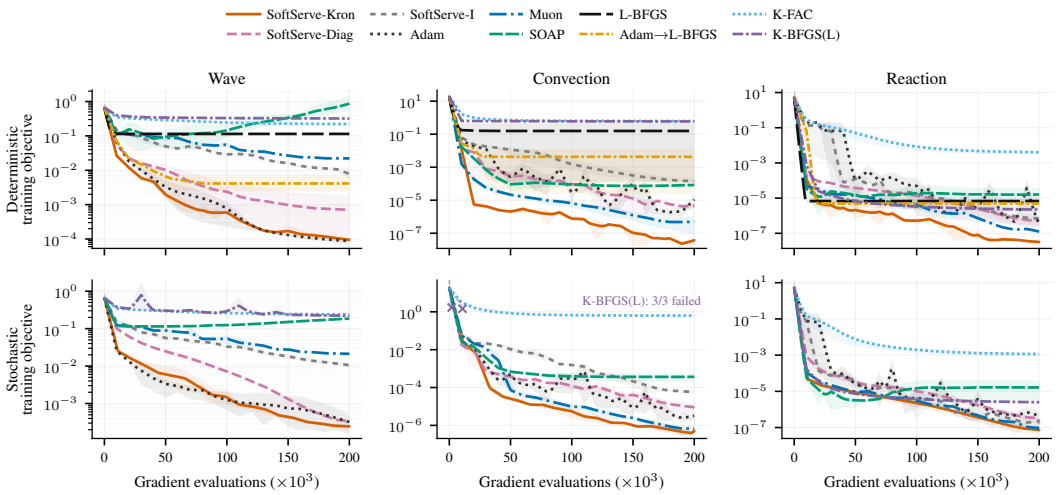


Figure 2: PINN training objectives for Wave, Convection, and Reaction, with fixed points (top) and resampled points (bottom). Lines show three-seed means; shading spans the seed minimum and maximum. Settings with failed seeds are shown as individual traces. Horizontal extensions show the final loss after an L-BFGS run terminates.

6.2 MNIST autoencoder

We next consider MNIST reconstruction, a standard benchmark for second-order optimization (Hinton & Salakhutdinov, 2006; Martens, 2010; Goldfarb et al., 2020). A 2.8M-parameter feed-forward autoencoder compresses each 28×28 image into a 30-dimensional representation and reconstructs the input. We minimize pixelwise binary cross-entropy with ℓ_2 regularization, using either all 60,000 training images at each update or minibatches of 1,000. These two settings compare deterministic and stochastic training of the same network.

SoftServe-Kron reaches the lowest training objective with full batches and is close to Muon with minibatches (Figure 1, middle, right). It improves on SoftServe-I in both settings. Appendix E.3 shows held-out reconstruction errors and wall-clock comparisons.

6.3 Physics-informed neural networks

Physics-informed neural networks (PINNs) approximate solutions to partial differential equations (PDEs) by mapping space–time coordinates to solution values. They are trained by penalizing the PDE residual and errors in the initial and boundary conditions. Depending on the differential operator, this loss can be very ill-conditioned, motivating the use of second-order optimization (Rathore et al., 2024). We evaluate SoftServe on both small feed-forward PINNs and the larger residual architecture PirateNet (Wang et al., 2024).

Small PINNs. We use the Wave, Convection, and Reaction problems of Rathore et al. (2024). Each network has three hidden layers of width 200 and 81,201 parameters. Training minimizes the sum of the mean squared residuals of the governing PDE and its associated initial and boundary conditions. We either fix the sampled training coordinates (collocation points) or resample them at each update. This gives six settings across the three equations.

SoftServe-Kron attains the lowest final training objective in five of the six settings, including all three with resampled points (Figure 2). On deterministic Wave, Adam and Adam+L-BFGS attain lower final losses, with Adam+L-BFGS achieving the lowest. Adam+L-BFGS, used by Rathore et al. (2024), runs Adam for the first 11,000 parameter updates, followed by L-BFGS within the remaining gradient-evaluation budget. Appendix E.4 gives solution errors and repeats the comparison in double precision.

For our remaining experiments with larger models below, we restrict the baselines to Adam, Muon, and SOAP, based on their scalability, and performance in the preceding experiments.

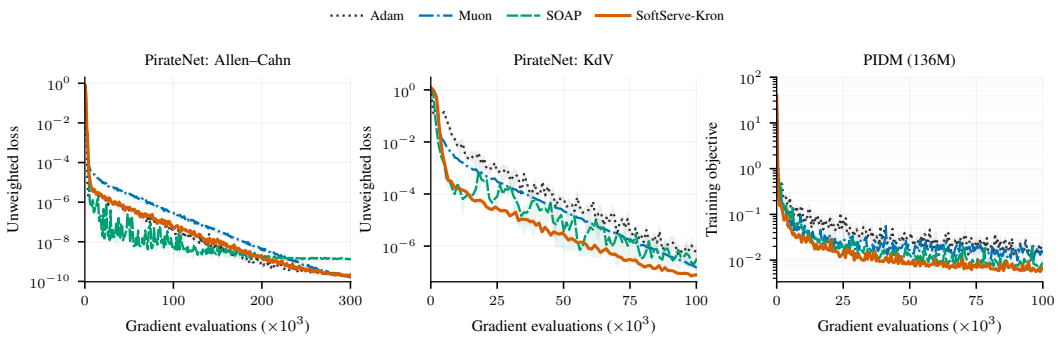


Figure 3: PirateNet on Allen–Cahn (left) and KdV (middle), and the 136M-parameter PIDM (right). PirateNet uses unweighted PDE and initial-condition MSE on fixed points; PIDM uses the training objective on fixed examples and noise. Lines show three-seed means; shading spans the seed minimum and maximum.

PirateNet. PirateNet is a PINN architecture that combines fully connected layers with skip connections to make deeper networks easier to train. We use a 0.73M-parameter model to solve the Allen–Cahn and Korteweg–de Vries (KdV) equations. Training uses 32,768 collocation points per update and adaptive weights for the loss terms. Since these weights differ across optimizers, we compare the unweighted sum of PDE and initial-condition MSE on the same fixed points. Against Adam, Muon, and SOAP, SoftServe-Kron is competitive with Muon on Allen–Cahn and reaches the lowest mean loss on KdV (Figure 3, left and middle). The PDEs and solution errors are given in Appendix E.5.

6.4 Physics-informed diffusion

To test SoftServe at a larger scale, we train the physics-informed diffusion model (PIDM) of Bastek et al. (2025) for structural design. Given applied forces, supports, and a material budget, the model learns to place material on a two-dimensional grid to form a stiff structure. We use the 136M-parameter U-Net and 30,000 optimized training structures. The network reconstructs material-density and displacement fields from noisy inputs. Its objective combines denoising error with penalties for force imbalance, material-volume error, and compliance, a measure of structural flexibility. At 100,000 gradient evaluations, SoftServe-Kron reaches a lower mean training loss than Adam, Muon, and SOAP (Figure 3, right).

6.5 Language model

Finally, we evaluate a 16M-parameter GPT-style language model (Radford et al., 2019). It has four layers, four attention heads per layer, hidden width 256, and a context length of 512 tokens. We minimize next-token cross-entropy on 200M tokens from FineWeb (Penedo et al., 2024). SoftServe-Kron reaches a mean validation NLL of 4.661, lower than AdamW (4.786) but higher than Muon (4.440) and SOAP (4.416). Thus, the gains on the preceding tasks do not extend to matching Muon and SOAP on this language model.

7 Discussion and Limitations

We introduce SoftServe, a quasi-Newton approach for deep learning. SoftServe considers structured Hessian approximations that are always positive definite, without line searches or curvature corrections, and it makes use of GPU-friendly Newton–Schulz approximations, allowing it to scale to massive neural networks. Our experiments highlight the value of curvature on several ill-conditioned tasks, including recurrent networks, autoencoders, and physics-informed models, including the 136M-parameter PIDM.

Many future directions remain. For instance, currently we use a fixed λ within each training run. Developing adaptive λ would further optimize performance. Finally, characterizing convergence guarantees under realistic settings remains an open problem to study.

AI USE STATEMENT

Generative AI tools assisted with drafting and editing prose and with plotting and analysis code. The authors take responsibility for the final content, including all AI-assisted text, code, and figures. The main ideas for this work come from our own 2024 internal draft, for which we did not use any AI tools. For writing, we mostly use AI for editing and suggesting re-writes. Whereas for code, we used AI more extensively, but we guarantee the code does as intended by a combination of reading the code ourselves and tests.

REPRODUCIBILITY STATEMENT

We provide a PyTorch implementation of SOFTSERVE, experiment runners, selected configurations, and data-preparation scripts at https://anonymous.4open.science/r/softserve-anonymous-602B. The appendix provides proofs and implementation details. Appendix E describes the datasets, architectures, hyperparameter selection, random seeds, and evaluation protocols.

REFERENCES

Noah Amsel, David Persson, Christopher Musco, and Robert M. Gower. The Polar Express: Optimal matrix sign methods and their application to the Muon algorithm. In *International Conference on Learning Representations (ICLR)*, 2026.

Jan-Hendrik Bastek, WaiChing Sun, and Dennis M. Kochmann. Physics-informed diffusion models. In *International Conference on Learning Representations (ICLR)*, 2025.

Albert S Berahas, Jorge Nocedal, and Martin Takac. A Multi-Batch L-BFGS Method for Machine Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Erik Berglund, Jiaojiao Zhang, and Mikael Johansson. Soft quasi-Newton: guaranteed positive definiteness by relaxing the secant constraint. *Optimization Methods and Software*, 40(4):783–812, 2025.

Antoine Bordes, Léon Bottou, and Patrick Gallinari. SGD-QN: Careful Quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research*, 10(59):1737–1754, 2009.

C. G. Broyden. The convergence of a class of double-rank minimization algorithms: 2. the new algorithm. *IMA Journal of Applied Mathematics*, 6(3):222–231, 1970.

Richard H. Byrd, S. L. Hansen, Jorge Nocedal, and Yoram Singer. A Stochastic Quasi-Newton Method for Large-Scale Optimization. *SIAM Journal on Optimization*, 26(2): 1008–1031, 2016.

Diana Cai, Chirag Modi, Loucas Pillaud-Vivien, Charles Margossian, Robert M. Gower, David Blei, and Lawrence K. Saul. Batch and match: black-box variational inference with a score-based divergence. In *International Conference on Machine Learning (ICML)*, 2024.

Michael Crawshaw, Chirag Modi, Mingrui Liu, and Robert M. Gower. An exploration of non-Euclidean gradient descent: Muon and its many variants. In *International Conference on Machine Learning (ICML)*, 2026.

J. E. Dennis, Jr. and Jorge J. Moré. Quasi-Newton Methods, Motivation and Theory. *SIAM Review*, 19(1):46–89, 1977.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61): 2121–2159, 2011.

R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3): 317–322, 1970.

R. Fletcher. A new variational result for Quasi-Newton formulae. *SIAM Journal on Optimization*, 1(1):18–21, 1991.

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-Newton methods for training deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning (ICML)*, 2018.

Nathan Heavner, Chao Chen, Abinand Gopal, and Per-Gunnar Martinsson. Efficient algorithms for computing rank-revealing factorizations on a GPU. *Numerical Linear Algebra with Applications*, 30(6):e2515, 2023.

Nicholas J. Higham. Stable iterations for the matrix square root. *Numerical Algorithms*, 15 (2):227–242, 1997.

Nicholas J Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, 2008.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge University Press, 2nd edition, 2012.

Brian Irwin and Eldad Haber. Secant penalized BFGS: a noise robust quasi-Newton method via penalizing the secant condition. *Computational Optimization and Applications*, 84(3): 651–702, 2023.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. Keller Jordan blog, 2024.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Peter Lancaster and Leiba Rodman. *Algebraic Riccati Equations*. Oxford University Press, 1995.

Xi-Lin Li. Preconditioned Stochastic Gradient Descent. *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):1454–1466, 2018.

Xi-Lin Li. Preconditioner on matrix Lie group for SGD. In *International Conference on Learning Representations (ICLR)*, 2019.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, et al. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025.

James Martens. Deep learning via Hessian-free optimization. In *International Conference on Machine Learning (ICML)*, 2010.

James Martens and Roger Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *International Conference on Machine Learning (ICML)*, 2015.

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

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2013.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained LMOs. In *International Conference on Machine Learning (ICML)*, 2025.

Omead Pooladzandi and Xi-Lin Li. Curvature-informed SGD via general purpose Lie-group preconditioners. *arXiv preprint arXiv:2402.04553*, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in training PINNs: A loss landscape perspective. In *International Conference on Machine Learning (ICML)*, 2024.

Yi Ren, Achraf Bahamou, and Donald Goldfarb. Kronecker-factored quasi-Newton methods for deep learning. *arXiv preprint arXiv:2102.06737v3*, 2022.

Nicol N. Schraudolph, Jin Yu, and Simon Günter. A Stochastic Quasi-Newton Method for Online Convex Optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

Günther Schulz. Iterative Berechnung der reziproken Matrix. *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 13(1):57–59, 1933.

D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. Coursera: Neural Networks for Machine Learning, 2012.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing Shampoo using Adam for language modeling. In *International Conference on Learning Representations (ICLR)*, 2025.

Sifan Wang, Bowen Li, Yuhan Chen, and Paris Perdikaris. PirateNets: Physics-informed deep learning with residual adaptive networks. *Journal of Machine Learning Research*, 25 (402):1–51, 2024.

Sifan Wang, Ananyae Kumar Bhartari, Bowen Li, and Paris Perdikaris. Gradient alignment in physics-informed neural networks: A second-order optimization perspective. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xiao Wang, Shiqian Ma, Donald Goldfarb, and Wei Liu. Stochastic Quasi-Newton Methods for Nonconvex Stochastic Optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017.

Philip Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969. doi: 10.1137/1011036.

Chenhui Xu, Dancheng Liu, Amir Nassereldine, and Jinjun Xiong. FP64 is all you need: Rethinking failure modes in physics-informed neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Appendix contents

A BFGS and Soft quasi-Newton **14**
A.1 BFGS . . . 14
A.2 Soft quasi-Newton . . . 14
A.3 Connections with Variational inference . . . 14

B SoftServe: structured updates **15**
B.1 Diagonal update . . . 15
B.2 Kronecker update . . . 15
B.3 Quadratic Matrix Equations . . . 17

C SoftServe: numerical computation of Kronecker updates **17**
C.1 Evaluating the updates in Gram form . . . 17
C.2 Newton-Schulz for the square root and inverse . . . 18
C.3 Computing the Kronecker updates via Newton-Schulz . . . 19

D Optimizer Details **19**
D.1 Training with SoftServe . . . 19
D.2 Routing and state . . . 22
D.3 Replay and gradient accounting . . . 22

E Experimental Details and Additional Results **22**
E.1 Experimental setting . . . 22
E.2 RNN Adding . . . 23
E.3 MNIST autoencoder . . . 25
E.4 Small physics-informed neural networks . . . 27
E.5 PirateNet . . . 32
E.6 Physics-informed diffusion . . . 35
E.7 Language model . . . 37

F Ablation studies **37**
F.1 Sensitivity to the secant penalty . . . 37
F.2 Accuracy of the QME solver . . . 38
F.3 Refresh frequency and training cost . . . 39

A BFGS and Soft quasi-Newton

A.1 BFGS

Typically, the BFGS update is derived via a weighted Frobenius norm objective that seeks to find the closest symmetric matrix to the current iterate's quasi-Newton matrix such that the secant equation is satisfied (Nocedal & Wright, 2006, Chapter 6). That is,

$$\boldsymbol{H}_{t+1} = \underset{\boldsymbol{H}\in\mathbb{S}_{++}}{\arg\min} \|\boldsymbol{H} - \boldsymbol{H}_t\|\, F(W) \quad \text{subject to} \quad \boldsymbol{H} = \boldsymbol{H}^\top, \quad \boldsymbol{H}y_t = s_t, \tag{A.1}$$

where the norm above is the weighted Frobenius norm $\|\boldsymbol{M}\|\, F(W) = \left\|\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{M}\boldsymbol{W}^{\frac{1}{2}}\right\|_{\mathrm{F}}$ with a weighting matrix $\boldsymbol{W}$ that satisfies the secant equation (e.g., $\boldsymbol{W} = \boldsymbol{B}_t$). The BFGS update is

$$\boldsymbol{H}_{t+1}^{\mathrm{BFGS}} = \left(I - \frac{s_t y_t^\top}{y_t^\top s_t}\right)\boldsymbol{H}_t\left(I - \frac{y_t s_t^\top}{y_t^\top s_t}\right) + \frac{s_t s_t^\top}{y_t^\top s_t}. \tag{A.2}$$

The log-determinant formulation of Fletcher (1991) given in Eq. 4 recovers the same update, providing the starting point for replacing the secant equation with a soft penalty.

A.2 Soft quasi-Newton

As discussed in the main text, Soft QN (Berglund et al., 2025) then considers the soft penalty, regularized by the log-determinant divergence: the Soft QN algorithm then approximates the inverse Hessian by solving

$$\boldsymbol{H}_{t+1} = \underset{\boldsymbol{H}\in\mathbb{S}^D_{++}}{\arg\min}\ \|s_t - \boldsymbol{H}y_t\|^2_{\boldsymbol{H}^{-1}} + \tfrac{1}{\lambda}\mathscr{D}(\boldsymbol{H}_t;\boldsymbol{H}). \tag{A.3}$$

Berglund et al. (2025) show that this objective has the unique, positive definite solution

$$\boldsymbol{H}_{t+1} = \boldsymbol{V}_t - \frac{4\lambda \boldsymbol{V}_t y_t y_t^\top \boldsymbol{V}_t}{\left(1 + \sqrt{1 + 4\lambda y_t^\top \boldsymbol{V}_t y_t}\right)^2}, \tag{A.4}$$

where $\boldsymbol{V}_t := \boldsymbol{H}_t + \lambda s_t s_t^\top$. Furthermore, Berglund et al. (2025, Theorem 3.2) shows that the BFGS update is recovered as a limiting case: under positive curvature $s_t^\top y_t > 0$, then

$$\boldsymbol{H}_{t+1} \to \boldsymbol{H}_{t+1}^{\mathrm{BFGS}}, \qquad \text{as } \lambda \to \infty. \tag{A.5}$$

A.3 Connections with Variational inference

For Gaussians with a common mean μ, the log-determinant divergence defined in Eq. 3 equals twice the KL divergence, that is

$$\mathscr{D}(\boldsymbol{P};\boldsymbol{Q}) = 2\,\mathrm{KL}(\mathcal{N}(\mu,\boldsymbol{P})\,\|\,\mathcal{N}(\mu,\boldsymbol{Q})).$$

Let $\pi = \mathcal{N}(\mu,\boldsymbol{H})$, $\pi_t = \mathcal{N}(\mu,\boldsymbol{H}_t)$, and let p be a differentiable positive density. With $s = \mu - z$ and $y = \nabla_z \log p(z)$, the Gaussian score is $\nabla_z \log \pi(z) = \boldsymbol{H}^{-1}s$, so

$$\|\nabla_z \log \pi(z) - y\|^2_{\boldsymbol{H}} = \|s - \boldsymbol{H}y\|^2_{\boldsymbol{H}^{-1}}, \qquad \tfrac{2}{\lambda}\mathrm{KL}(\pi_t\|\pi) = \tfrac{1}{\lambda}\mathscr{D}(\boldsymbol{H}_t;\boldsymbol{H}). \tag{A.6}$$

Thus, the objective in Eq. A.3 is the one-score covariance objective of Cai et al. (2024) with the mean fixed. The identity links the two covariance subproblems; it does not equate variational inference with Quasi-Newton optimization.

B SoftServe: structured updates

Recall that SoftServe solves the soft quasi-Newton objective with respect to a structured matrix family $\mathcal{H}$ (diagonal or Kronecker factored):

$$\boldsymbol{H}_{t+1} = \arg\min_{\boldsymbol{H}\in\mathcal{H}} \ \|s_t - \boldsymbol{H}y_t\|^2_{\boldsymbol{H}^{-1}} + \tfrac{1}{\lambda}\mathscr{D}(\boldsymbol{H}_t;\boldsymbol{H}). \tag{B.1}$$

B.1 Diagonal update

We begin by considering the diagonal family

$$\mathcal{H}_{\text{Diag}} = \{\text{diag}(h) : h \in \mathbb{R}^D_{++}\}.$$

The lemma below derives the update.

Lemma B.1 (Exact diagonal update)**.** Let $h_t \in \mathbb{R}^D_{++}$, $\boldsymbol{H}_t = \text{diag}(h_t)$, $s_t, y_t \in \mathbb{R}^D$, and $\lambda > 0$. The diagonal restriction of Eq. B.1 has the unique minimizer $\boldsymbol{H}_{t+1} = \text{diag}(h_{t+1})$, where

$$h_{t+1,i} = \frac{2\left(h_{t,i} + \lambda s^2_{t,i}\right)}{1 + \sqrt{1 + 4\lambda y^2_{t,i}\left(h_{t,i} + \lambda s^2_{t,i}\right)}}, \qquad i = 1, \dots, D. \tag{B.2}$$

Hence $h_{t+1} \in \mathbb{R}^D_{++}$, regardless of the signs of $s_{t,i}y_{t,i}$.

Proof. For $\boldsymbol{H} = \text{diag}(h)$, the terms of Eq. B.1 that depend on h are

$$\sum_{i=1}^{D}\left[\frac{s^2_{t,i}}{h_i} + h_i y^2_{t,i} + \frac{1}{\lambda}\left(\frac{h_{t,i}}{h_i} + \log h_i\right)\right].$$

Fix i, set $v = h_{t,i} + \lambda s^2_{t,i} > 0$ and $b = h_i^{-1}$, and multiply by λ. The scalar problem is

$$\underset{b>0}{\text{minimize}} \quad \psi(b), \qquad \psi(b) = vb + \tfrac{\lambda y^2_{t,i}}{b} - \log b.$$

Since $\psi''(b) = 2\lambda y^2_{t,i}/b^3 + 1/b^2 > 0$ and $\psi(b) \to +\infty$ as $b \downarrow 0$ or $b \to \infty$, the minimizer exists and is unique. The stationarity condition $\lambda y^2_{t,i} h_i^2 + h_i - v = 0$ has the single positive root

$$h_i = \frac{2v}{1 + \sqrt{1 + 4\lambda y^2_{t,i} v}},$$

which also covers $y_{t,i} = 0$. Separability over i proves the claim. □

B.2 Kronecker update

We state standard identities for Kronecker products (Horn & Johnson, 2012).

Lemma B.2. For $\boldsymbol{P} \in \mathbb{R}^{m\times m}$, $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, and conformable $\boldsymbol{X}, \boldsymbol{Z}$, we have:

$$(\boldsymbol{P}_1 \otimes \boldsymbol{Q}_1)(\boldsymbol{P}_2 \otimes \boldsymbol{Q}_2) = (\boldsymbol{P}_1\boldsymbol{P}_2) \otimes (\boldsymbol{Q}_1\boldsymbol{Q}_2) \tag{B.3}$$

$$(\boldsymbol{P} \otimes \boldsymbol{Q})\,\text{vec}(\boldsymbol{X}) = \text{vec}(\boldsymbol{Q}\boldsymbol{X}\boldsymbol{P}^\top) \tag{B.4}$$

$$\text{vec}(\boldsymbol{X})^\top \text{vec}(\boldsymbol{Z}) = \text{tr}(\boldsymbol{X}^\top\boldsymbol{Z}) \tag{B.5}$$

$$\text{tr}(\boldsymbol{P} \otimes \boldsymbol{Q}) = \text{tr}(\boldsymbol{P})\,\text{tr}(\boldsymbol{Q}) \tag{B.6}$$

$$\det(\boldsymbol{P} \otimes \boldsymbol{Q}) = \det(\boldsymbol{P})^n \det(\boldsymbol{Q})^m. \tag{B.7}$$

We now consider the Kronecker-structured family:

$$\mathcal{H}_{\text{Kron}} = \{\boldsymbol{L} \otimes \boldsymbol{R} : \boldsymbol{L} \in \mathbb{S}^m_{++}, \boldsymbol{R} \in \mathbb{S}^n_{++}\}. \tag{B.8}$$

Here, we derive the alternating update from Section 3.2. First, we recall some notation. Write $s_t = \text{vec}(\boldsymbol{S}_t)$, $y_t = \text{vec}(\boldsymbol{Y}_t)$, and $\boldsymbol{H}_t = \boldsymbol{L}_t \otimes \boldsymbol{R}_t$. The following result establishes the properties of these conditional updates.

Lemma B.3 (Alternating Kronecker solution.)**.** Let $\boldsymbol{L}_t \in \mathbb{S}^m_{++}$, $\boldsymbol{R}_t \in \mathbb{S}^n_{++}$, $\lambda > 0$, and $\boldsymbol{S}_t, \boldsymbol{Y}_t \in \mathbb{R}^{n\times m}$. Consider the objective given in Eq. 5 and the updates

$$\boldsymbol{L}_{t+1} = \underset{\boldsymbol{L}\in\mathbb{S}^m_{++}}{\arg\min} \quad \text{tr}(\boldsymbol{U}_L\boldsymbol{L}) + \text{tr}(\boldsymbol{V}_L\boldsymbol{L}^{-1}) + \log\det\boldsymbol{L} \tag{B.9}$$

$$\boldsymbol{R}_{t+1} = \underset{\boldsymbol{R}\in\mathbb{S}^n_{++}}{\arg\min} \quad \text{tr}(\boldsymbol{U}_R\boldsymbol{R}) + \text{tr}(\boldsymbol{V}_R\boldsymbol{R}^{-1}) + \log\det\boldsymbol{R}, \tag{B.10}$$

where we define the auxiliary matrices:

$$\boldsymbol{U}_L := \tfrac{\lambda}{n}\boldsymbol{Y}_t^\top\boldsymbol{R}_t\boldsymbol{Y}_t \quad \text{and} \quad \boldsymbol{V}_L := \boldsymbol{L}_t + \tfrac{\lambda}{n}\boldsymbol{S}_t^\top\boldsymbol{R}_t^{-1}\boldsymbol{S}_t \tag{B.11}$$

$$\boldsymbol{U}_R := \tfrac{\lambda}{m}\boldsymbol{Y}_t\boldsymbol{L}_{t+1}\boldsymbol{Y}_t^\top \quad \text{and} \quad \boldsymbol{V}_R := m^{-1}[\text{tr}(\boldsymbol{L}_t\boldsymbol{L}_{t+1}^{-1})\boldsymbol{R}_t + \lambda\boldsymbol{S}_t\boldsymbol{L}_{t+1}^{-1}\boldsymbol{S}_t^\top]. \tag{B.12}$$

These updates have a unique, closed-form solution that costs $\mathcal{O}(n^3 + m^3 + n^2m + nm^2)$ to compute exactly. Furthermore, the updates to $\boldsymbol{L}$ and $\boldsymbol{R}$ monotonically decrease the soft QN objective.

Proof. Substituting $\boldsymbol{H} = \boldsymbol{L} \otimes \boldsymbol{R}$ into the objective in Eq. B.1 and applying trace and determinant identities gives, up to a constant

$$\begin{aligned}\Phi_t(\boldsymbol{L}, \boldsymbol{R}) = \ & \text{tr}(\boldsymbol{L}\boldsymbol{Y}_t^\top\boldsymbol{R}\boldsymbol{Y}_t) + \text{tr}(\boldsymbol{L}^{-1}\boldsymbol{S}_t^\top\boldsymbol{R}^{-1}\boldsymbol{S}_t) \\ & + \tfrac{1}{\lambda}\,\text{tr}(\boldsymbol{L}_t\boldsymbol{L}^{-1})\,\text{tr}(\boldsymbol{R}_t\boldsymbol{R}^{-1}) + \tfrac{1}{\lambda}\left[n\log\det\boldsymbol{L} + m\log\det\boldsymbol{R}\right].\end{aligned} \tag{B.13}$$

This expression makes the dependence on each factor explicit. We minimize it first over $\boldsymbol{L}$ with $\boldsymbol{R} = R_t$, then over $\boldsymbol{R}$ with $\boldsymbol{L} = L_{t+1}$ fixed.

Holding $\boldsymbol{R} = \boldsymbol{R}_t$ fixed and collecting terms in Eq. 5 involving $\boldsymbol{L}$, we obtain the conditional update

$$\boldsymbol{L}_{t+1} = \underset{\boldsymbol{L}\in\mathbb{S}^m_{++}}{\arg\min} \quad \text{tr}(\boldsymbol{U}_L\boldsymbol{L}) + \text{tr}(\boldsymbol{V}_L\boldsymbol{L}^{-1}) + \log\det\boldsymbol{L}. \tag{B.14}$$

Differentiating with respect to $\boldsymbol{L}$ and setting the derivative to zero, yields the quadratic matrix equation

$$\boldsymbol{L}\,\boldsymbol{U}_L\,\boldsymbol{L} + \boldsymbol{L} = \boldsymbol{V}_L. \tag{B.15}$$

Here, because $\boldsymbol{U}_L \succeq 0$ and $\boldsymbol{V}_L \succ 0$, this equation has a unique, positive-definite solution $\boldsymbol{L}_{t+1}$ (Lemma B.4).

We then update the factor $\boldsymbol{R}$ in an analogous manner, holding $\boldsymbol{L}_{t+1}$ fixed:

$$\boldsymbol{R}_{t+1} = \underset{\boldsymbol{R}\in\mathbb{S}^n_{++}}{\arg\min} \quad \text{tr}(\boldsymbol{U}_R\boldsymbol{R}) + \text{tr}(\boldsymbol{V}_R\boldsymbol{R}^{-1}) + \log\det\boldsymbol{R}, \tag{B.16}$$

Solving Eq. B.16 for $\boldsymbol{R}_{t+1}$ reduces to solving another quadratic matrix equation

$$\boldsymbol{R}\,\boldsymbol{U}_R\,\boldsymbol{R} + \boldsymbol{R} = \boldsymbol{V}_R, \tag{B.17}$$

which again has a unique, positive-definite solution.

Each block update minimizes the objective over one factor, holding the other factor and the reference matrix $\boldsymbol{H}_t$ fixed. Since the current factor is feasible, each update cannot increase the objective. Forming the coefficient products costs $\mathcal{O}(n^2m + nm^2)$; the factor inverses and two QME solves cost $\mathcal{O}(n^3 + m^3)$. □

B.3 Quadratic Matrix Equations

Recall that SoftServe-Kron has an alternating update for the factors $\boldsymbol{L}$ and $\boldsymbol{R}$, where first we perform

$$\text{Update } \boldsymbol{L}_{t+1} \text{ by solving } \boldsymbol{L}\boldsymbol{U}_L\boldsymbol{L} + \boldsymbol{L} = \boldsymbol{V}_L, \tag{B.18}$$

and then we perform the update

$$\text{Update } \boldsymbol{R}_{t+1} \text{ by solving } \boldsymbol{R}\boldsymbol{U}_R\boldsymbol{R} + \boldsymbol{R} = \boldsymbol{V}_R. \tag{B.19}$$

Given known matrices $\boldsymbol{U} \succeq 0$ and $\boldsymbol{V} \succ 0$, each update above involves solving a quadratic matrix equation of the form

$$\boldsymbol{Q}\boldsymbol{U}\boldsymbol{Q} + \boldsymbol{Q} = \boldsymbol{V}. \tag{B.20}$$

Cai et al. (2024, Lemma B.2) show that Eq. B.21 below is the positive definite solution. Furthermore, due to strict convexity of the objective function in $\boldsymbol{Q}^{-1}$, from which the quadratic matrix equation was defined, this solution is unique.

Lemma B.4 (Quadratic matrix equation)**.** The positive definite solution to Eq. B.20 is given by

$$\boldsymbol{Q} = 2\boldsymbol{V}^{1/2}\left[\boldsymbol{I}_d + \left(\boldsymbol{I}_d + 4\boldsymbol{V}^{1/2}\boldsymbol{U}\boldsymbol{V}^{1/2}\right)^{1/2}\right]^{-1}\boldsymbol{V}^{1/2}. \tag{B.21}$$

When updating either Kronecker factor, $\boldsymbol{L}$ or $\boldsymbol{R}$, the matrix $\boldsymbol{U}$ in Eq. B.20 admits a Gram factorization $\boldsymbol{U} = \boldsymbol{F}\boldsymbol{F}^\top$. Cai et al. (2024, Lemma B.3) give the equivalent solution to the quadratic matrix equation in terms of the Gram matrix

$$\boldsymbol{Q} = \boldsymbol{V} - \boldsymbol{V}^\top\boldsymbol{F}\left[\tfrac{1}{2}\boldsymbol{I} + (\boldsymbol{F}^\top\boldsymbol{V}\boldsymbol{F} + \tfrac{1}{4}\boldsymbol{I})^{\frac{1}{2}}\right]^{-2}\boldsymbol{F}^\top\boldsymbol{V}. \tag{B.22}$$

For a matrix $\boldsymbol{F} \in \mathbb{R}^{d\times p}$, this update takes $\mathcal{O}(d^2)$ computation.

Our implementation of SoftServe-Kron, which relies on the coupled Newton-Schulz, will rely on the latter Gram-based solution in Eq. B.22.

C SoftServe: numerical computation of Kronecker updates

The Kronecker updates in Section 3.2 require solving two quadratic matrix equations, one for each factor. In this section, we describe how to evaluate these updates using matrix multiplications and Newton-Schulz iterations, as outlined in Section 4.1. We begin by explaining how to form the coefficients $\boldsymbol{U}, \boldsymbol{V}$ of the quadratic matrix equations, then we describe the Newton-Schulz procedures, and finally we explain how to apply the Newton-Schulz procedures to obtain the full Gram update, providing explicit pseudocode for the Kronecker factor updates $\boldsymbol{L}$ and $\boldsymbol{R}$.

C.1 Evaluating the updates in Gram form

In this section, we show that the coefficients $\boldsymbol{U} = \boldsymbol{F}\boldsymbol{F}^\top$ and $\boldsymbol{V}$ can be written in terms of matrix multiplications and square root and inverse square root operations. Consider the Gram solution to the quadratic matrix equation in Eq. B.22. This expression requires a square root of $\boldsymbol{C} = \boldsymbol{I} + 4\boldsymbol{F}^\top\boldsymbol{V}\boldsymbol{F}$ and an inverse of $\boldsymbol{I} + \boldsymbol{C}^{\frac{1}{2}}$. Overall, this solution requires fewer matrix function evaluations (i.e., square root and inverse) than the solution in Eq. B.21.

Algorithm C.1 Newton–Schulz routines

function NewtonSchulzRootPair($\boldsymbol{M}, J$)
  $a \leftarrow \|\boldsymbol{M}\|_F$
  $\boldsymbol{Z} \leftarrow \boldsymbol{M}/a$
  $\boldsymbol{P} \leftarrow \boldsymbol{I}$
  for $j = 1, \ldots, J$ **do**
    $\boldsymbol{T} \leftarrow \frac{1}{2}(3\boldsymbol{I} - \boldsymbol{P}\boldsymbol{Z})$
    $\boldsymbol{Z} \leftarrow \mathrm{sym}[\boldsymbol{Z}\boldsymbol{T}]$
    $\boldsymbol{P} \leftarrow \mathrm{sym}[\boldsymbol{T}\boldsymbol{P}]$
  return $(\sqrt{a}\,\boldsymbol{Z}; \boldsymbol{P}/\sqrt{a})$ ▷ Approximations to $\boldsymbol{M}^{\frac{1}{2}}, \boldsymbol{M}^{-\frac{1}{2}}$

function NewtonSchulzInverse($\boldsymbol{M}, J$)
  $\boldsymbol{E} \leftarrow \boldsymbol{I}/\|\boldsymbol{M}\|_F$
  for $j = 1, \ldots, J$ **do**
    $\boldsymbol{E} \leftarrow \mathrm{sym}[\boldsymbol{E}(2\boldsymbol{I} - \boldsymbol{M}\boldsymbol{E})]$
  return $\boldsymbol{E}$ ▷ Approximation to $\boldsymbol{M}^{-1}$

We begin by motivating the Gram update further by examining the factorization $\boldsymbol{U} = \boldsymbol{F}\boldsymbol{F}^\top$. For our Kronecker factor updates, we first examine the definitions of $\boldsymbol{U}_L$ and $\boldsymbol{U}_R$:

$$\boldsymbol{U}_L = \tfrac{\lambda}{n}\boldsymbol{Y}_t^\top \boldsymbol{R}_t \boldsymbol{Y}_t \tag{C.1}$$

$$\boldsymbol{U}_R = \tfrac{\lambda}{m}\boldsymbol{Y}_t \boldsymbol{L}_{t+1} \boldsymbol{Y}_t^\top. \tag{C.2}$$

Here, we observe that they are already in the desired form, and so we do not need to factor the matrices $\boldsymbol{U}_L$ and $\boldsymbol{U}_R$. Thus, the required factorizations arise from

$$\boldsymbol{U}_L = \boldsymbol{F}_L \boldsymbol{F}_L^\top, \qquad \boldsymbol{F}_L = \sqrt{\tfrac{\lambda}{n}}\boldsymbol{Y}_t^\top \boldsymbol{R}_t^{\frac{1}{2}} \tag{C.3}$$

$$\boldsymbol{U}_R = \boldsymbol{F}_R \boldsymbol{F}_R^\top, \qquad \boldsymbol{F}_R = \sqrt{\tfrac{\lambda}{m}}\boldsymbol{Y}_t \boldsymbol{L}_{t+1}^{\frac{1}{2}}. \tag{C.4}$$

The remaining coefficients $\boldsymbol{V}_L$ and $\boldsymbol{V}_R$ can be formed with the *corresponding* inverse square roots to the square roots used above. For example, for $\boldsymbol{V}_L = \boldsymbol{L}_t + \frac{\lambda}{n}\boldsymbol{S}_t^\top \boldsymbol{R}_t^{-1} \boldsymbol{S}_t$, we can form the second term using

$$\boldsymbol{S}_t^\top \boldsymbol{R}_t^{-1} \boldsymbol{S}_t = (\boldsymbol{R}_t^{-\frac{1}{2}} \boldsymbol{S}_t)^\top (\boldsymbol{R}_t^{-\frac{1}{2}} \boldsymbol{S}_t). \tag{C.5}$$

This construction allows us to reuse both outputs of each coupled Newton-Schulz call: the square root forms $\boldsymbol{F}_L$ (or $\boldsymbol{F}_R$), and the corresponding inverse square root forms $\boldsymbol{V}_L$ (or $\boldsymbol{V}_R$), avoiding a separate Newton-Schulz call to approximate the inverse $\boldsymbol{R}_t^{-1}$.

In Algorithm C.2, we summarize how to form all of the coefficients in this manner.

C.2 Newton-Schulz for the square root and inverse

To compute the Gram solution of the quadratic matrix equation, we need an approximation for the matrix square root and the matrix inverse. We approximate the required matrix functions using Newton-Schulz iterations, which construct polynomial approximations through repeated matrix multiplications and additions. We summarize these routines in Algorithm C.1 as NewtonSchulzRootPair and NewtonSchulzInverse, respectively.

For square roots, we use the coupled Newton-Schulz iteration, which simultaneously approximates a matrix square root and its inverse using matrix multiplications (Higham, 1997; 2008). For an input matrix $\boldsymbol{M} \in \mathbb{S}_{++}^d$, let $\|\boldsymbol{M}\|_F$ denote its Frobenius norm. First, we rescale the matrix $\|\boldsymbol{M}\|_F$, placing the eigenvalues between 0 and 1:

$$a = \|\boldsymbol{M}\|_F, \tag{C.6}$$

$$\boldsymbol{Z}_0 = \boldsymbol{M}/a, \tag{C.7}$$

$$\boldsymbol{P}_0 = \boldsymbol{I}. \tag{C.8}$$

In the Newton-Schulz procedure, we then iterate $j = 0, \ldots, J-1$ times through

$$\boldsymbol{T}_j = \tfrac{1}{2}(3\boldsymbol{I} - \boldsymbol{P}_j \boldsymbol{Z}_j) \tag{C.9}$$
$$\boldsymbol{Z}_{j+1} = \mathrm{sym}[\boldsymbol{Z}_j \boldsymbol{T}_j] \tag{C.10}$$
$$\boldsymbol{P}_{j+1} = \mathrm{sym}[\boldsymbol{T}_j \boldsymbol{P}_j], \tag{C.11}$$

where we define the matrix symmetrization operation

$$\mathrm{sym}[\boldsymbol{A}] = (\boldsymbol{A} + \boldsymbol{A}^\top)/2. \tag{C.12}$$

The iterates $\boldsymbol{Z}_j$ and $\boldsymbol{P}_j$ approximate the square root and inverse square root, respectively, of the scaled input. The correction $\boldsymbol{T}_j = \frac{1}{2}(3\boldsymbol{I} - \boldsymbol{P}_j \boldsymbol{Z}_j)$ is a first-degree polynomial in $\boldsymbol{P}_j \boldsymbol{Z}_j$, approximating its inverse square root around the identity.

After J iterations, rescaling by $\sqrt{a}$ gives the desired approximations with respect to the original input matrix

$$\boldsymbol{M}^{\frac{1}{2}} \approx \sqrt{a}\, \boldsymbol{Z}_J \tag{C.13}$$
$$\boldsymbol{M}^{-\frac{1}{2}} \approx \boldsymbol{P}_J / \sqrt{a}. \tag{C.14}$$

To approximate the inverse of the matrix, we apply a separate Newton-Schulz iteration (Schulz, 1933; Higham, 2008). For a positive definite matrix $\boldsymbol{M} \in \mathbb{S}^d_{++}$, we initialize and then iterate $j = 0, \ldots, J-1$ times

$$\boldsymbol{E}_0 = \boldsymbol{I}/\|\boldsymbol{M}\|_F \tag{C.15}$$
$$\boldsymbol{E}_{j+1} = \mathrm{sym}[\boldsymbol{E}_j(2\boldsymbol{I} - \boldsymbol{M}\boldsymbol{E}_j)]. \tag{C.16}$$

This update builds a polynomial approximation to $\boldsymbol{M}^{-1}$. After J iterations, we return the approximation $\boldsymbol{E}_J \approx \boldsymbol{M}^{-1}$.

C.3 Computing the Kronecker updates via Newton-Schulz

We now combine the Newton-Schulz routines to evaluate the Gram update in Eq. B.22. Given $\boldsymbol{F}$ and $\boldsymbol{V}$, we form $\boldsymbol{C} = \boldsymbol{I} + 4\boldsymbol{F}^\top \boldsymbol{V} \boldsymbol{F}$ and apply NewtonSchulzRootPair to obtain an approximation $\widehat{\boldsymbol{B}} \approx \boldsymbol{C}^{\frac{1}{2}}$. We then apply NewtonSchulzInverse to obtain $\widehat{\boldsymbol{W}} \approx (\boldsymbol{I} + \widehat{\boldsymbol{B}})^{-1}$. The resulting Newton-Schulz approximation to the Gram solution of the quadratic matrix equation is then given by

$$\widehat{\boldsymbol{Q}} = \boldsymbol{V} - 4(\boldsymbol{V}\boldsymbol{F}\widehat{\boldsymbol{W}})(\boldsymbol{V}\boldsymbol{F}\widehat{\boldsymbol{W}})^\top. \tag{C.17}$$

The resulting update only requires these two Newton-Schulz routines and matrix multiplications. Algorithm C.2 presents the pseudocode for this Gram update and its application within the alternating Kronecker update for the factors $\boldsymbol{L}$ and $\boldsymbol{R}$. In our implementation, we use 18 iterations for the square root routines, and 10 for the inverse routine.

In the application of these procedures to the quadratic matrix equation solution, our implementation symmetrizes every output and, for every solve, records the normalized residual

$$\rho_Q = \|\boldsymbol{Q}\boldsymbol{U}\boldsymbol{Q} + \boldsymbol{Q} - \boldsymbol{V}\|_{\mathrm{F}} \,/\, \|\boldsymbol{V}\|_{\mathrm{F}}\,.$$

D Optimizer Details

D.1 Training with SoftServe

We provide more of the details of how our SoftServe methods are implemented. In particular, this section will explain the detailed pseudo code of SoftServe-Kron and SoftServe-Diag is in Algorithm D.1 and Algorithm D.2, respectively.

SoftServe-Kron is a method for updating matrix parameters, where we will use $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ to denote the matrix parameters of a given layer. Whereas SoftServe-Diag is applicable to

Algorithm C.2 Kronecker updates using the Gram solution

function SolveGram($\boldsymbol{F}, \boldsymbol{V}, J_{\text{root}}, J_{\text{inv}}$) ▷ Approximately solve $\boldsymbol{Q}\boldsymbol{F}\boldsymbol{F}^\top\boldsymbol{Q} + \boldsymbol{Q} = \boldsymbol{V}$
  $\boldsymbol{C} \leftarrow \boldsymbol{I} + 4\boldsymbol{F}^\top\boldsymbol{V}\boldsymbol{F}$
  $(\boldsymbol{B}, _) \leftarrow$ NewtonSchulzRootPair($\boldsymbol{C}, J_{\text{root}}$)
  $\boldsymbol{W} \leftarrow$ NewtonSchulzInverse($\boldsymbol{I} + \boldsymbol{B}, J_{\text{inv}}$)
  $\boldsymbol{E} \leftarrow \boldsymbol{V}\boldsymbol{F}\boldsymbol{W}$
  return sym$[\boldsymbol{V} - 4\boldsymbol{E}\boldsymbol{E}^\top]$

function KroneckerUpdate($\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{S}, \boldsymbol{Y}, \lambda, J_{\text{root}}, J_{\text{inv}}$)
▷ Update $\boldsymbol{L}$, holding $\boldsymbol{R}$ fixed
  $(\boldsymbol{C}_R, \boldsymbol{D}_R) \leftarrow$NewtonSchulzRootPair($\boldsymbol{R}, J_{\text{root}}$)
  $\boldsymbol{F}_L \leftarrow \sqrt{\frac{\lambda}{n}}\,\boldsymbol{Y}^\top\boldsymbol{C}_R$ ▷ Form factored coefficient $\boldsymbol{U}_L = \boldsymbol{F}_L\boldsymbol{F}_L^\top$
  $\boldsymbol{Z}_R \leftarrow \boldsymbol{D}_R\boldsymbol{S}$
  $\boldsymbol{V}_L \leftarrow L + \frac{\lambda}{n}\,\boldsymbol{Z}_R^\top\,\boldsymbol{Z}_R$ ▷ Form coefficient $\boldsymbol{V}_L$
  $\boldsymbol{L}^+ \leftarrow$ SolveGram($\boldsymbol{F}_L, \boldsymbol{V}_L, J_{\text{root}}, J_{\text{inv}}$)
▷ Update $\boldsymbol{R}$, holding $\boldsymbol{L}^+$ fixed
  $(\boldsymbol{C}_L, \boldsymbol{D}_L) \leftarrow$NewtonSchulzRootPair($\boldsymbol{L}^+, J_{\text{root}}$)
  $\boldsymbol{F}_R \leftarrow \sqrt{\frac{\lambda}{m}}\,\boldsymbol{Y}\boldsymbol{C}_L$ ▷ Form factored coefficient $\boldsymbol{U}_R = \boldsymbol{F}_R\boldsymbol{F}_R^\top$
  $\boldsymbol{Z}_L \leftarrow \boldsymbol{S}\boldsymbol{D}_L$
  $\alpha_L \leftarrow \frac{1}{m}\,\text{tr}(\boldsymbol{D}_L\boldsymbol{L}\boldsymbol{D}_L)$ ▷ Depends on original factor $\boldsymbol{L}$
  $\boldsymbol{V}_R \leftarrow \alpha_L\boldsymbol{R} + \frac{\lambda}{m}\,\boldsymbol{Z}_L\boldsymbol{Z}_L^\top$ ▷ Form coefficient $\boldsymbol{V}_R$
  $\boldsymbol{R}^+ \leftarrow$ SolveGram($\boldsymbol{F}_R, \boldsymbol{V}_R, J_{\text{root}}, J_{\text{inv}}$)
return $(\boldsymbol{L}^+, \boldsymbol{R}^+)$

parameters of any shape, and thus in Algorithm D.2 we represent the given parameters as a vector $x \in \mathbb{R}^d$. If this corresponds to a linear layer, then $d = nm$, where $n \times m$ are the dimensions of the linear layer.

To make SoftServe practical, we will make use of some now-standard practices in deep learning, including the use of momentum, updating the approximate Hessian every so often to amortize costs, and taking care to properly scale the update directions. We address each of these considerations in turn.

To estimate the gradient for the quasi-Newton step, we will use momentum estimate which in matrix form is updated according to lines (6–7) in Algorithm D.1, and in vector form is updated according to lines (6–7) in Algorithm D.2. We also incorporate an additional Nesterov extrapolation or the debiasing trick Kingma & Ba (2015) to form the final momentum estimate.

Restricting the Hessian approximation to a diagonal or Kronecker factor greatly reduces the costs as compared to the dense Soft QN method. For instance, for a $n \times m$ linear layer, the state, application, and refresh costs of SoftServe are given in Table D.1. In particular, the cost for applying and refreshing the Kronecker-factored update in SoftServe-Kron is still cubic in the dimensions of the layer. To amortize this cost, we will update our estimate of the inverse Hessian only once every $K \in \mathbb{N}$ iterations. That is, once every K iterations we take a snapshot of the current iterate, gradient, and sample in line 5 in Algorithm D.1. We then use this snap shot to form our difference of iterates $\boldsymbol{S}$ and gradients $\boldsymbol{Y}$ on lines 10 and 11. The analogous snapshot for and its application for SoftServe-Diag are on lines 5, 10 and 11 in Algorithm D.2.

Metric	State	Apply	Refresh
Dense	$\mathcal{O}(n^2m^2)$	$\mathcal{O}(n^2m^2)$	$\mathcal{O}(n^2m^2)$
Diag	$\mathcal{O}(nm)$	$\mathcal{O}(nm)$	$\mathcal{O}(nm)$
Kron, $n \times m$	$\mathcal{O}(n^2 + m^2)$	$\mathcal{O}(n^2m + nm^2)$	$\mathcal{O}(n^3 + m^3 + n^2m + nm^2)$

Table D.1: SoftServe method costs for updating the parameters of a $n \times m$ linear layer.

Algorithm D.1 SOFTSERVE-Kron: detailed implementation

Require: Step size γ_t; Number of iterations $T \in \mathbb{N}$, Refresh rate $K \in [T]$, Penalty λ; momentum parameters $\beta \in (0,1)$, Nesterov $\in \{True, False\}$

1: **Initialize** $\boldsymbol{X}_0 \in \mathbb{R}^{n\times m}$, $\boldsymbol{M}_{-1} = 0$, $\boldsymbol{L}_0 = \boldsymbol{I}_m$ and $\boldsymbol{R}_0 = \boldsymbol{I}_n$.
2: **for** $t = 0, \dots, T-1$ **do**
3: Draw ξ_t and evaluate $\boldsymbol{G}_t \leftarrow \nabla f_{\xi_t}(\boldsymbol{X}_t)$
4: **if** $t \bmod K = 0$ **then**
5: $\bar{\xi} \leftarrow \xi_t,\ \bar{\boldsymbol{X}} \leftarrow \boldsymbol{X}_t,\ \bar{\boldsymbol{G}} \leftarrow \boldsymbol{G}_t$ ▷ Take snapshot
6: $\boldsymbol{M}_t = \beta \boldsymbol{M}_{t-1} + (1-\beta)\boldsymbol{G}_t$ ▷ Update Momentum buffer
7: $\hat{\boldsymbol{M}}_t = \begin{cases} (1-\beta)\boldsymbol{G}_t + \beta \boldsymbol{M}_t & \text{if Nesterov} = \texttt{True} \\ \boldsymbol{M}_t/(1-\beta^{t+1}) & \text{if Nesterov} = \texttt{False} \end{cases}$ ▷ Use bias correction or Nesterov
8: $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t - \gamma_t \frac{\boldsymbol{R}_t \hat{\boldsymbol{M}}_t \boldsymbol{L}_t}{\sqrt{\operatorname{tr}\left(\hat{\boldsymbol{M}}_t^\top \boldsymbol{R}_t \hat{\boldsymbol{M}}_t \boldsymbol{L}_t\right)}}$ ▷ Update parameters
9: **if** $(t+1) \bmod K = 0$ and $t+1 < T$ **then** ▷ Update Hessian estimate
10: $\boldsymbol{S} \leftarrow \boldsymbol{X}_{t+1} - \bar{\boldsymbol{X}}$
11: $\boldsymbol{Y} \leftarrow \nabla f_{\bar{\xi}}(\boldsymbol{X}_{t+1}) - \bar{\boldsymbol{G}}$
12: Update Kronecker factors by Gram Newton-Schulz (Algorithm C.2) to solve

$$\text{Update } \boldsymbol{L}_{t+1} \text{ by solving } \boldsymbol{L}\boldsymbol{U}_L\boldsymbol{L} + \boldsymbol{L} = \boldsymbol{V}_L$$
$$\text{Update } \boldsymbol{R}_{t+1} \text{ by solving } \boldsymbol{R}\boldsymbol{U}_R\boldsymbol{R} + \boldsymbol{R} = \boldsymbol{V}_R,$$

where $\boldsymbol{U}_L, \boldsymbol{V}_L$ are given in Eq. 9, and $\boldsymbol{U}_R, \boldsymbol{V}_R$ are given in Eq. 12.
13: Rebalance the Kronecker factors

$$(\boldsymbol{L}_{t+1}, \boldsymbol{R}_{t+1}) = (\boldsymbol{L}_{t+1}/c,\ c\boldsymbol{R}_{t+1}) \qquad \text{where } c = \sqrt{\frac{\operatorname{tr}(\boldsymbol{L}_{t+1})/m}{\operatorname{tr}(\boldsymbol{R}_{t+1})/n}}. \tag{D.1}$$

14: **return** $\boldsymbol{X}_T$

The differences of iterates and gradients are then used to update the factorized and diagonal Hessian estimate on line 12 in both Algorithm D.1 and Algorithm D.2, respectively. To be more efficient (or *pythonic*) we have written the update of the diagonal estimate h_{t+1} line 12 in Algorithm D.2 as a vector operation as opposed to an element-wise operation as introduced in Eq. 6.

Algorithm D.2 SOFTSERVE-Diag detailed implementation

Require: Step size γ_t; Number of iterations $T \in \mathbb{N}$, Refresh rate $K \in [T]$, Penalty λ; momentum parameters $\beta \in (0,1)$, Nesterov $\in \{True, False\}$

1: **Initialize** $x_0 \in \mathbb{R}^d$, $m_{-1} \leftarrow 0$ and $h_0 = (1, \dots, 1) \in \mathbb{R}^d$.
2: **for** $t = 0, \dots, T-1$ **do**
3: Draw ξ_t and evaluate $g_t \leftarrow \nabla f_{\xi_t}(x_t)$
4: **if** $t \bmod K = 0$ **then**
5: $\bar{\xi} \leftarrow \xi_t,\ \bar{x} \leftarrow x_t,\ \bar{g} \leftarrow g_t$ ▷ Take snapshot
6: $m_t = \beta m_{t-1} + (1-\beta) g_t$
7: $\hat{m}_t = \begin{cases} (1-\beta)g_t + \beta m_t & \text{if Nesterov} = \texttt{True} \\ m_t/(1-\beta^{t+1}) & \text{if Nesterov} = \texttt{False} \end{cases}$
8: $x_{t+1} = x_t - \gamma_t \dfrac{h_t \odot \hat{m}_t}{\sqrt{\hat{m}_t^\top (h_t \odot \hat{m}_t)}}$ ▷ Update parameters
9: **if** $(t+1) \bmod K = 0$ and $t+1 < T$ **then** ▷ Update Hessian estimate
10: $s \leftarrow x_{t+1} - \bar{x}$
11: $y \leftarrow \nabla f_{\bar{\xi}}(x_{t+1}) - \bar{g}$
12: $h_{t+1} = 2\,(h_t + \lambda\, s \odot s) \oslash \left[\mathbf{1} + \sqrt{\mathbf{1} + 4\lambda\,(y \odot y) \odot (h_t + \lambda\, s \odot s)}\right]$
13: **return** x_T

Our final modification to make SOFTSERVE practical for deep learning is to normalize the update. We will explain this modification for a vector parameter $x \in \mathbb{R}^d$. The same modification for matrix parameters follows by vectorizing, and then re-stacking as a matrix.

Traditionally quasi-Newton methods are applied according to

$$x_{t+1} = x_t - \gamma_t \boldsymbol{H}_t \hat{m}_t. \tag{D.2}$$

This update Eq. D.2 corresponds to the *regularized* descent method:

$$x_{t+1} = \arg\min_{x \in \mathbb{R}^d} \ \hat{m}_t^\top (x - x_t) + \frac{1}{2\gamma_t} \|x - x_t\|^2_{\boldsymbol{H}_t^{-1}}.$$

But now there is a growing body of evidence that the *constrained* descent method (Jordan et al., 2024; Pethick et al., 2025; Crawshaw et al., 2026) given by

$$\begin{aligned} x_{t+1} &= \arg\min_{x \in \mathbb{R}^d} \ \hat{m}_t^\top (x - x_t) \quad \text{subject to } \|x - x_t\|_{\boldsymbol{H}_t^{-1}} \le \gamma_t \\ &= x_t - \gamma_t \boldsymbol{H}_t \hat{m}_t / \sqrt{\hat{m}_t^\top \boldsymbol{H}_t \hat{m}_t} \end{aligned} \tag{D.3}$$

generally works better for deep learning. Our initial experiments also confirmed that the constrained descent Eq. D.3 resulting in better methods and thus we used Eq. D.3 throughout our experiments to update the parameters, see line 8 in Algorithm D.1 and Algorithm D.2.

In these updates, note that the product $\boldsymbol{H}_t \hat{m}_t$ in Eq. D.3 is given by

$$\boldsymbol{H}_t \hat{m}_t = \begin{cases} \mathrm{vec}\big(\boldsymbol{R}_t \hat{\boldsymbol{M}}_t \boldsymbol{L}_t\big) & \text{in line 8 in Algorithm D.1 ,} \\ h_t \odot \hat{m}_t & \text{in line 8 in Algorithm D.2 ,} \end{cases} \tag{D.4}$$

where $\odot$ is the elementwise (Hadamard) product.

D.2 Routing and state

The parameter partition is disjoint. A Diag block stores one positive value per coordinate; a Kron block for an $n \times m$ matrix stores $\boldsymbol{L} \in \mathbb{S}^m_{++}$ and $\boldsymbol{R} \in \mathbb{S}^n_{++}$. Blocks without a matrix interpretation are routed to Diag, or to a first-order fallback in the neural experiments (Appendix E). Kron factors of equal shape are stacked for batched matrix multiplication. The initial states are $h_0 = \mathbf{1}$, the all-ones vector, and $(\boldsymbol{L}_0, \boldsymbol{R}_0) = (\boldsymbol{I}_m, \boldsymbol{I}_n)$. The metric is fixed within each interval of K parameter steps. Each refresh uses one raw interval pair; we do not average pairs or interpolate metric states.

D.3 Replay and gradient accounting

A replay key stores every random choice that affects f_{ζ_t}: the minibatch indices, augmentation, dropout masks, and any other sampled loss input. Replaying the key at both endpoints removes sampling mismatch from y_t. The replayed gradient is used only for the refresh; it does not update momentum, the learning-rate schedule, or the parameters. For N parameter steps, Algorithm D.1 evaluates N ordinary gradients and $\lfloor (N-1)/K \rfloor$ replayed ones, and we count both. With decoupled weight decay, the displacement s_t is the actual parameter displacement, including the decay. A checkpoint taken inside an interval must retain the momentum, the metric state, the interval start, the cached raw gradient, and the replay key.

E Experimental Details and Additional Results

E.1 Experimental setting

Hardware and arithmetic. We use NVIDIA RTX PRO 6000 Blackwell GPUs for training. Models use FP32, with BF16 autocasting for GPT.

Selection and evaluation. We tune learning rates on seed 0 using logarithmic grids, extending a grid when its minimum lies at the boundary. Final runs use seeds 1–3. Within each task, methods share initialization seeds and data. Test errors are not used for selection. Curves show three-seed means and variability bands; settings with failed seeds are shown as individual traces.

RNN Adding, MNIST, and the small PINNs screen the first-moment coefficient β_1 for SoftServe, SoftServe-I, Adam, Muon, and SOAP. K-FAC and K-BFGS(L) also receive a momentum search. In these studies, Nesterov on/off is tested for SoftServe and Muon; Adam and SOAP retain the momentum update rules of their implementations while their β_1 values are tuned. The task descriptions below specify which settings are tuned or fixed.

Optimizer settings. SoftServe uses $K = 10$, and one alternating factor sweep per refresh. Kron uses the Gram QME implementation in Algorithm C.2, with 18 coupled root iterations and 10 inverse iterations; Diag uses the closed-form elementwise update.

Gradient accounting and plots. Gradient budgets count training gradients, secant replays, line-search evaluations, and loss-balancing gradients. In stochastic training, replay adds approximately 10% to SoftServe-Kron's count at $K = 10$, so it takes fewer parameter updates within the same budget.

E.2 RNN Adding

Task and data. Each length-T input has $T = 100$ pairs $z_\tau = (a_\tau, m_\tau)$, where $a_\tau \sim \text{Uniform}[0, 1]$. Exactly two markers m_τ equal one, one in each half of the sequence. The target is the sum of the marked numbers. Every update draws 128 new sequences; validation and test sets each contain 8,192 fixed sequences per seed.

Model and objective. The network uses $h_\tau = \tanh(W_h h_{\tau-1} + W_z z_\tau + b)$, hidden width 16, $h_0 = 0$, and a scalar affine readout, for 321 parameters. The target and batch loss are

$$y = \sum_{\tau=1}^{T} m_\tau a_\tau, \qquad \mathcal{L}(\theta) = \frac{1}{B} \sum_{i=1}^{B} \big(\widehat{y}_\theta(z^{(i)}) - y^{(i)}\big)^2. \tag{E.1}$$

Here B is the batch size and $\widehat{y}_\theta$ is the prediction.

Training and tuning. The budget is 40,000 gradient evaluations. An 8,000-gradient pilot screens hyperparameters, followed by full-budget learning-rate sweeps selected by validation MSE (Figure E.1). The selected values are $\lambda = 999$ for SoftServe-Kron and $\lambda = 1$ for SoftServe-Diag.

Evaluation. We report minibatch training MSE and fixed-set validation and test MSE. Figure E.2 shows training and validation MSE against parameter updates and wall-clock time. K-BFGS(L)'s two failed seeds are shown individually.

Local conditioning. We compute the full 321×321 loss Hessian at initialization on a fixed batch of 128 sequences for each of seeds 0–3, using FP64 differentiation. All four Hessians are indefinite. Each has one to six singular values below $321\varepsilon_{64}\sigma_{\max}$, where $\varepsilon_{64} = 2^{-52}$ and $\sigma_{\max}$ is the largest singular value. Thus, these sampled losses are numerically nearly singular at initialization; a finite condition number cannot be resolved reliably (Figure E.3).

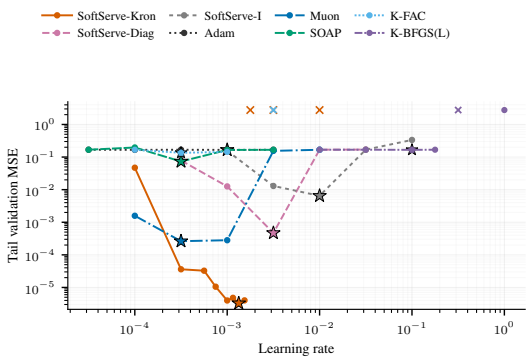


Figure E.1: RNN Adding learning-rate sweeps on seed 0. Stars mark selected rates; crosses mark numerical failures.

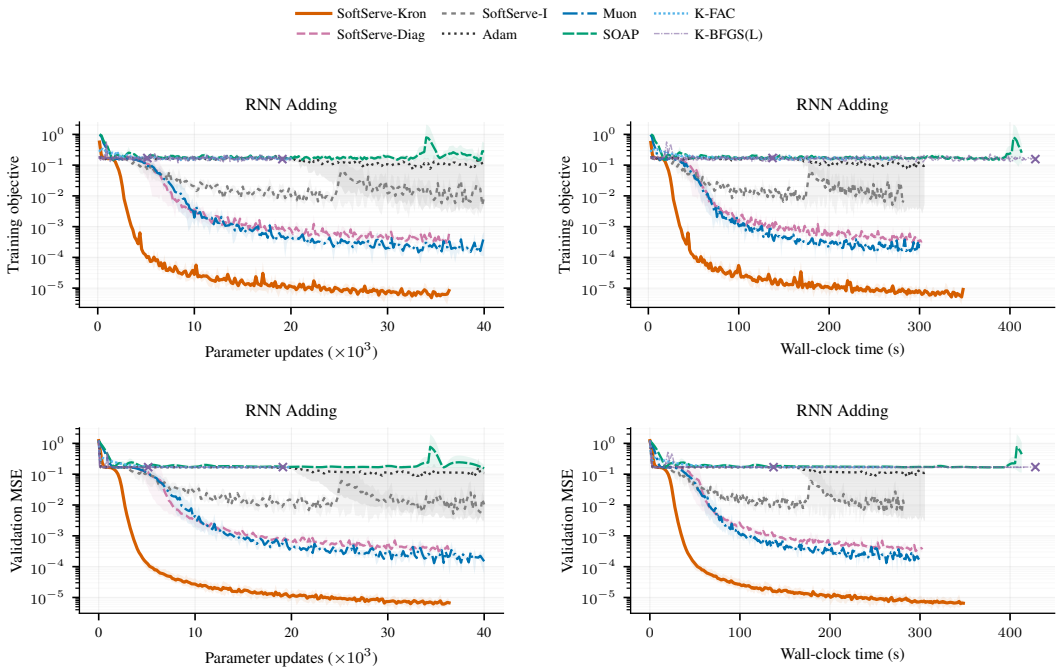


Figure E.2: RNN Adding training minibatch MSE and held-out validation MSE against parameter updates and wall-clock time. Failed cohorts retain their individual traces.

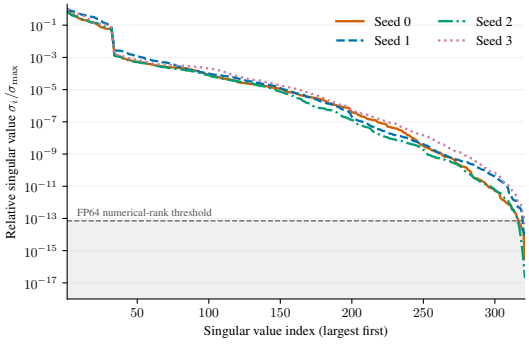


Figure E.3: RNN Adding loss-Hessian singular values at initialization, normalized by the largest value. The dashed line marks the numerical-rank threshold $321\varepsilon_{64}$. All four seeds have singular values below this threshold.

E.3 MNIST AUTOENCODER

Task and data. We use MNIST, comprising 28×28 grayscale images of handwritten digits, with 60,000 training and 10,000 test examples and pixel values in $[0, 1]$. The task is to compress each image into a low-dimensional representation and reconstruct the original input, without using digit labels. This autoencoder task is a standard benchmark for second-order optimization (Goldfarb et al., 2020).

Model and objective. We use a 2.8M-parameter fully connected autoencoder with a symmetric encoder–decoder architecture and a 30-dimensional linear bottleneck. The remaining hidden layers use ReLU, and the output uses sigmoid. Following Goldfarb et al. (2020), the batch objective is

$$\mathcal{L}(\theta) = -\frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{784}\left[x_{ij}\log\widehat{x}_{ij} + (1-x_{ij})\log(1-\widehat{x}_{ij})\right] + \frac{10^{-5}}{2}\|\theta\|_2^2, \tag{E.2}$$

where B is the batch size, x is the input image, and $\widehat{x}$ is its reconstruction.

Training and tuning. Full-batch training uses 4,000 gradient evaluations; minibatch training uses batches of 1,000 images and 6,000 evaluations. SOFTSERVE-Kron, SOFTSERVE-I, Muon, and SOAP use Adam on biases, while SOFTSERVE-Diag acts on all parameters. Full-batch L-BFGS uses a strong-Wolfe line search.

A quarter-budget pilot screens hyperparameters, followed by full-budget learning-rate sweeps selected by the mean of the final five training objectives (Figure E.4). We select $\lambda = 999$ for SOFTSERVE-Kron and $\lambda = 9$ for SOFTSERVE-Diag.

Evaluation. Figure E.5 reports test reconstruction BCE, excluding the training regularizer. Figure E.6 shows the regularized training objective against parameter updates and wall-clock time. Failed runs are shown individually rather than included in three-seed means.

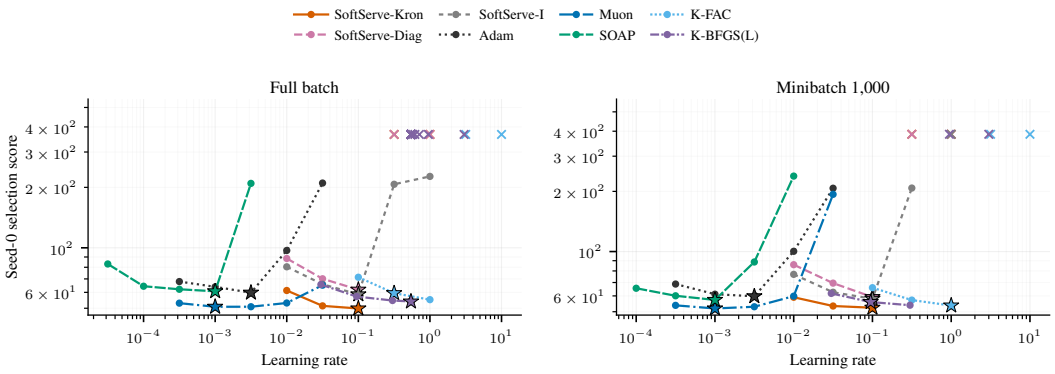


Figure E.4: Autoencoder full-budget LR sweeps at selected momentum and fixed λ. Stars mark selected rates. Crosses mark failed or off-scale runs.

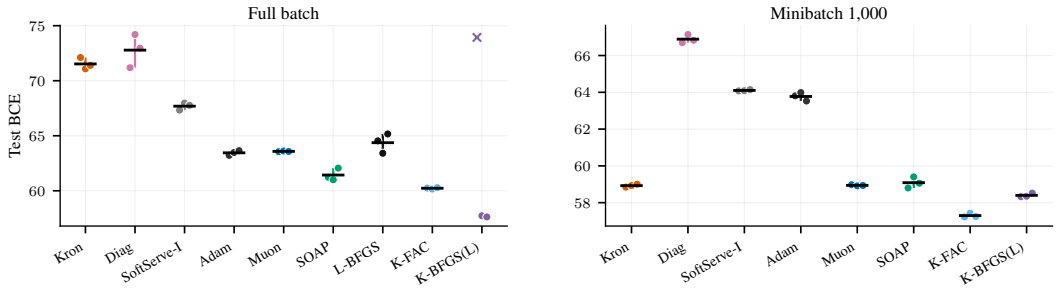


Figure E.5: Autoencoder test reconstruction BCE. Dots are individual seeds; ticks are three-seed means. Full-batch K-BFGS(L) has one failed seed and therefore no three-seed mean.

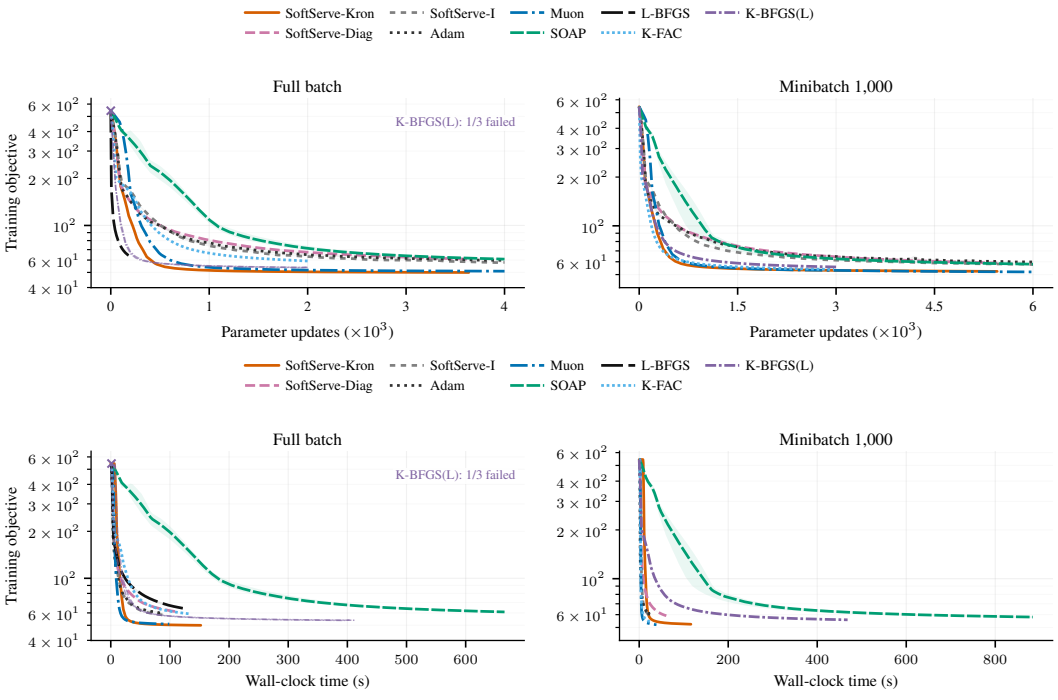


Figure E.6: Autoencoder training objective against parameter updates and wall-clock time. Shading spans the three seed values.

E.4 Small physics-informed neural networks

Task and data. We consider three one-dimensional problems from Rathore et al. (2024). The scalar function $u(x,t)$ denotes the solution at spatial coordinate x and time t. All problems use $t \in [0,1]$.

Convection. On the spatial domain $[0, 2\pi]$, we solve

$$\begin{aligned} \frac{\partial u}{\partial t} + 40\frac{\partial u}{\partial x} &= 0, \\ u(x,0) &= \sin x, \\ u(0,t) &= u(2\pi,t). \end{aligned}$$

The analytical solution is

$$u^\star(x,t) = \sin(x - 40t).$$

Reaction. On the spatial domain $[0, 2\pi]$, we solve

$$\begin{aligned} \frac{\partial u}{\partial t} - 5u(1-u) &= 0, \\ u(x,0) &= h(x), \\ u(0,t) &= u(2\pi,t), \end{aligned}$$

where $h(x) = \exp[-(x-\pi)^2/(2(\pi/4)^2)]$. The analytical solution is

$$u^\star(x,t) = \frac{h(x)e^{5t}}{h(x)e^{5t} + 1 - h(x)}.$$

Wave. On the spatial domain $[0,1]$, we solve

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - 4\frac{\partial^2 u}{\partial x^2} &= 0, \\ u(x,0) &= \sin(\pi x) + \tfrac{1}{2}\sin(5\pi x), \\ \frac{\partial u}{\partial t}(x,0) &= 0, \\ u(0,t) = u(1,t) &= 0. \end{aligned}$$

The analytical solution is

$$u^\star(x,t) = \sin(\pi x)\cos(2\pi t) + \tfrac{1}{2}\sin(5\pi x)\cos(10\pi t).$$

The analytical solutions are used to evaluate solution errors, not as interior training targets.

Model and objective. We use a fully connected network with three width-200 tanh hidden layers and 81,201 parameters to predict $u_\theta(x,t)$. The training objective is

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BC}}.$$

The PDE term is the mean squared equation residual. The initial- and boundary-condition terms sum the MSEs for the prescribed conditions. For Wave, these include both initial displacement and velocity, and the values at both spatial boundaries.

Training and tuning. We use 200,000 gradient evaluations and cosine learning-rate decay to one tenth of the initial rate. In the deterministic setting, the 10,000 interior collocation points remain fixed throughout training. In the stochastic setting, they are resampled at every update. Both gradients of each secant pair are evaluated on the same points.

Following hyperparameter screening, we select learning rates by the mean of the final five training losses (Figure E.7). SoftServe-Kron uses $\beta_1 = 0.995$. For Wave, Convection, and Reaction, respectively, its selected λ values are $(99, 9999, 999)$ in the deterministic setting and 99 for all three in the stochastic setting. In the same problem order, SoftServe-Diag uses $(9,1,1)$ for deterministic training and $(9,1,9)$ for stochastic training.

SoftServe-Kron, SoftServe-I, Muon, and SOAP update the two 200×200 hidden weight matrices, with Adam on the remaining parameters. Deterministic experiments also include L-BFGS and Adam followed by L-BFGS, using strong-Wolfe line searches.

Evaluation. We measure relative solution error,

$$\frac{\|u_\theta - u^\star\|_2}{\|u^\star\|_2},$$

with both solutions evaluated on a 129×101 space–time grid. Figure E.8 reports FP32 solution errors. Figures E.11 and E.12 show training losses against parameter updates and wall-clock time.

Since numerical precision can affect PINN convergence (Xu et al., 2025), we repeat the experiments in FP64 using the FP32-selected configurations without retuning. Figures E.9 and E.10 report the corresponding training losses and solution errors. Failed runs are shown individually. When an L-BFGS line search returns zero displacement, we extend its final value horizontally to indicate that the iterate remains unchanged.

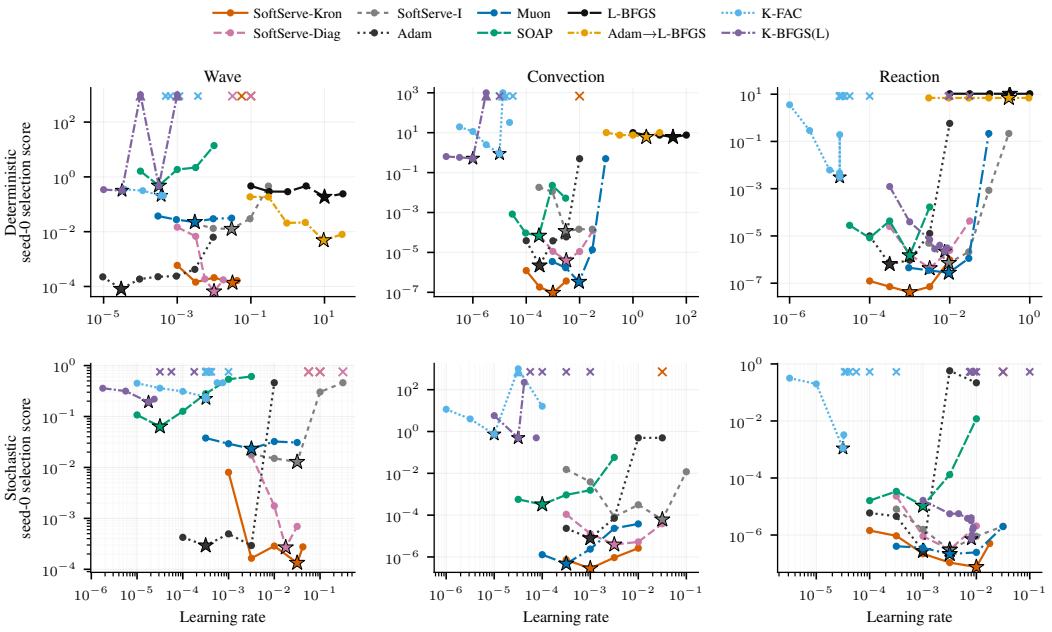


Figure E.7: Small PINN learning-rate sweeps on seed 0, scored by the mean of the final five training losses. Stars mark selected rates, crosses numerical failures, and triangles off-scale finite scores.

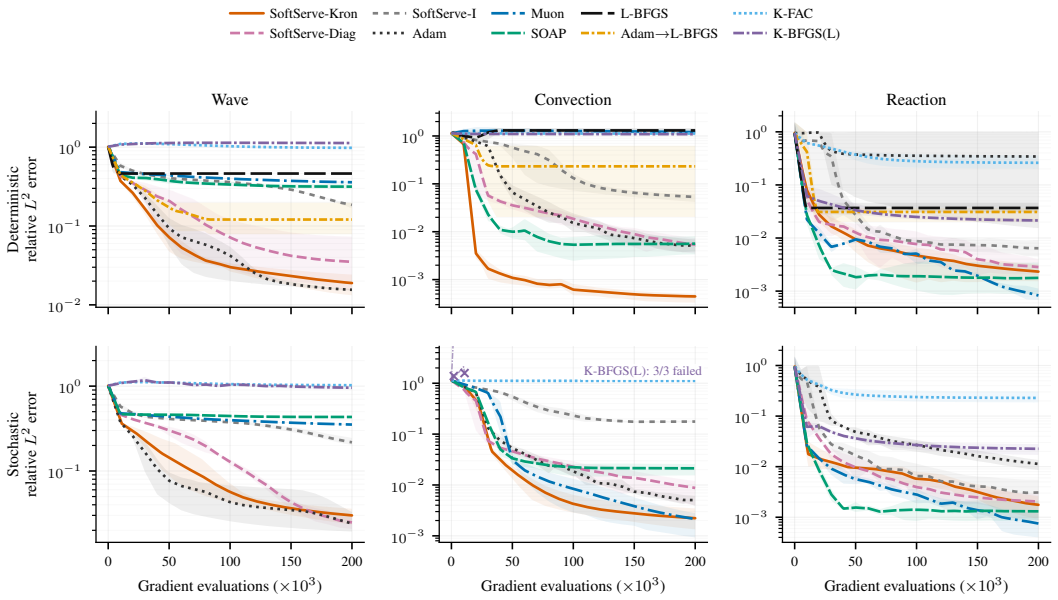


Figure E.8: FP32 PINN relative solution error. Columns: Wave, Convection, Reaction. Rows: fixed and resampled collocation points. The horizontal continuation of a terminated L-BFGS run represents its unchanged solution.

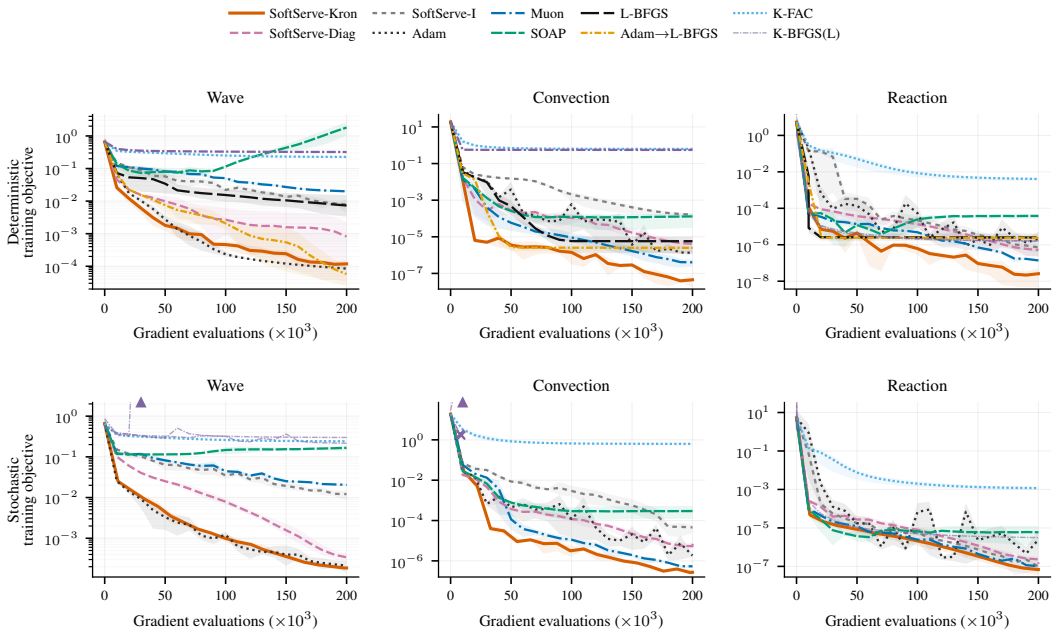


Figure E.9: FP64 PINN training objectives with the FP32-selected settings. The gradient budget includes replay and line-search evaluations.

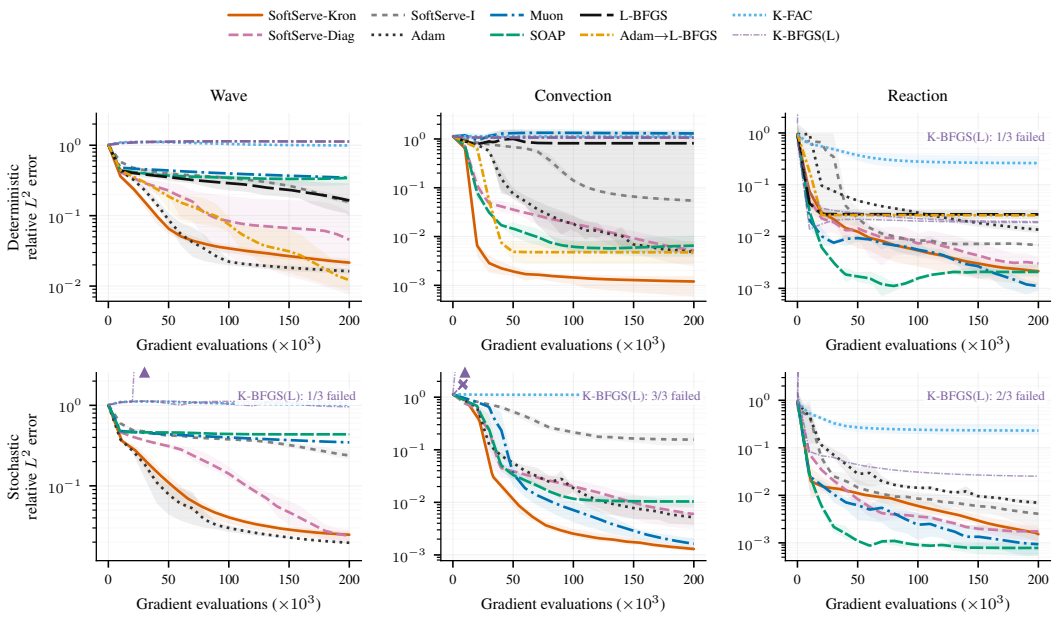


Figure E.10: FP64 PINN solution errors with the FP32-selected settings. Settings with failed seeds are shown as individual traces.

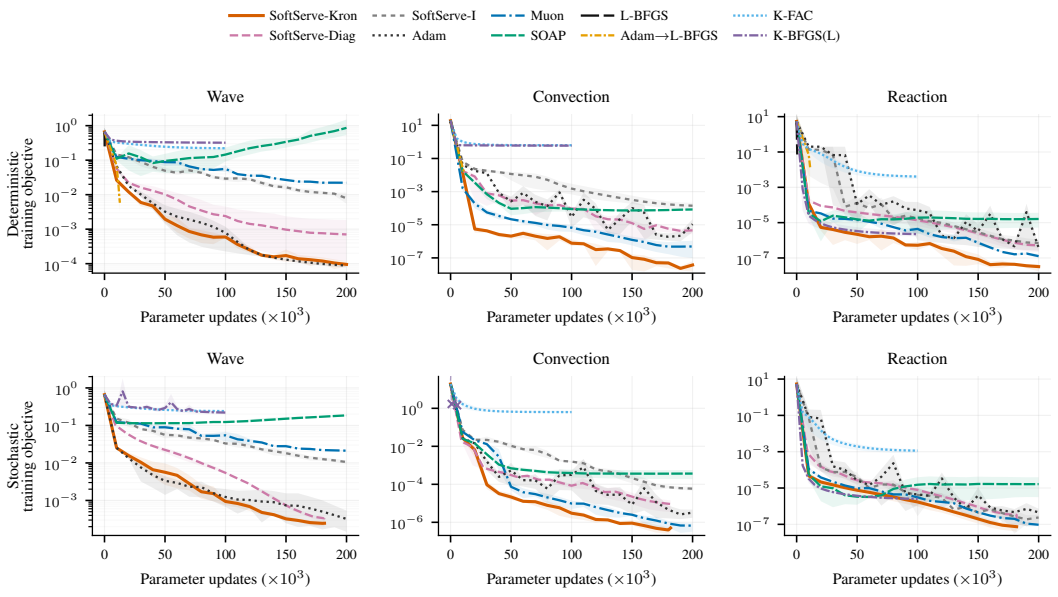


Figure E.11: FP32 PINN training objectives against parameter updates.

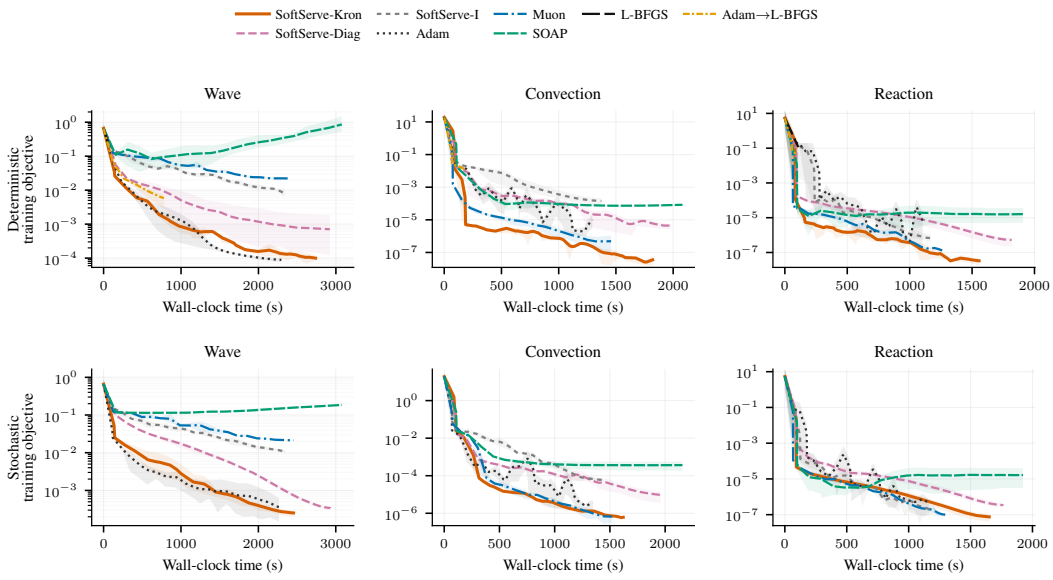


Figure E.12: FP32 PINN training objectives against wall-clock time. K-FAC and K-BFGS(L) are omitted here because their timers also include construction time.

E.5 PirateNet

Task and data. We solve the Allen–Cahn and Korteweg–de Vries (KdV) equations on $(x, t) \in [-1, 1] \times [0, 1]$, where $u(x, t)$ is the scalar solution.

Allen–Cahn.

$$\frac{\partial u}{\partial t} - 10^{-4}\frac{\partial^2 u}{\partial x^2} + 5u^3 - 5u = 0,$$
$$u(x, 0) = x^2 \cos(\pi x).$$

KdV.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + 0.022^2\frac{\partial^3 u}{\partial x^3} = 0,$$
$$u(x, 0) = \cos(\pi x).$$

Both problems use periodic spatial boundaries. Reference solutions for evaluation are taken from the `jaxpi` repository.[1]

Model and objective. We use PirateNet (Wang et al., 2024), a residual PINN with three width-256 blocks, tanh activations, and approximately 0.73M parameters. The network uses periodic spatial features, and its final layer is initialized by least squares against the initial condition.

Following Wang et al. (2025), training minimizes a weighted combination of PDE residual and initial-condition MSEs. Causal weighting prioritizes earlier times, while gradient-norm balancing adjusts the relative weights of the loss terms. These weights are treated as constants during differentiation and held fixed between the two secant evaluations.

Training and tuning. Each update samples 32,768 uniform collocation points. We compare methods at 300,600 gradient evaluations for Allen–Cahn and 100,200 for KdV, counting both secant replay and loss-balancing gradients. Learning rates use a 5,000-update warmup followed by exponential decay. Matrix methods use $\beta_1 = 0.995$ and Adam on the remaining parameters. Momentum settings are fixed during tuning.

Learning-rate sweeps run for 300,000 updates on Allen–Cahn and 100,000 on KdV, with selection based on unweighted validation loss (Figure E.13). The selected λ values for SoftServe-Kron are $\lambda = 9999$ for Allen–Cahn and $\lambda = 999$ for KdV.

Evaluation. Although all optimizers use the same weighting rules, their training weights evolve differently. We therefore compare the unweighted sum of PDE and initial-condition MSEs on common fixed points. Figure E.14 shows this loss against parameter updates and wall-clock time.

Figure E.15 reports relative L^2 solution errors on the reference grid. Solution errors are not used for learning-rate selection. All curves stop at the same gradient budgets as the main figures, so SoftServe-Kron takes fewer parameter updates because of replay.

[1] https://github.com/PredictiveIntelligenceLab/jaxpi

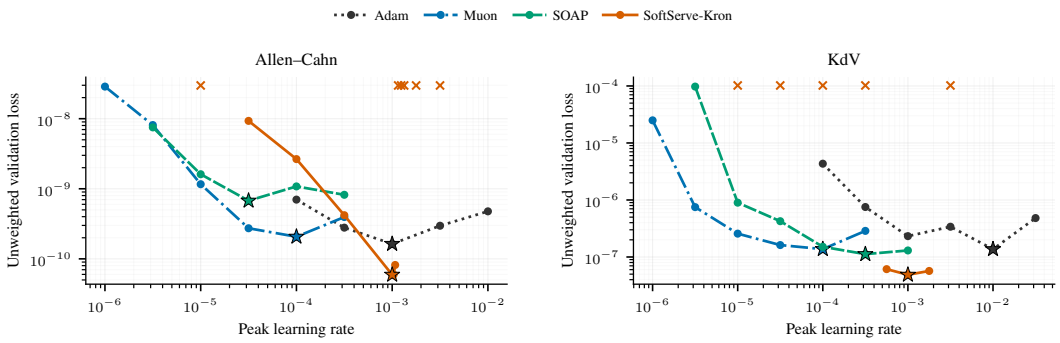


Figure E.13: PirateNet full-budget learning-rate sweeps, scored by unweighted validation loss. Stars mark selected rates; crosses mark failures.

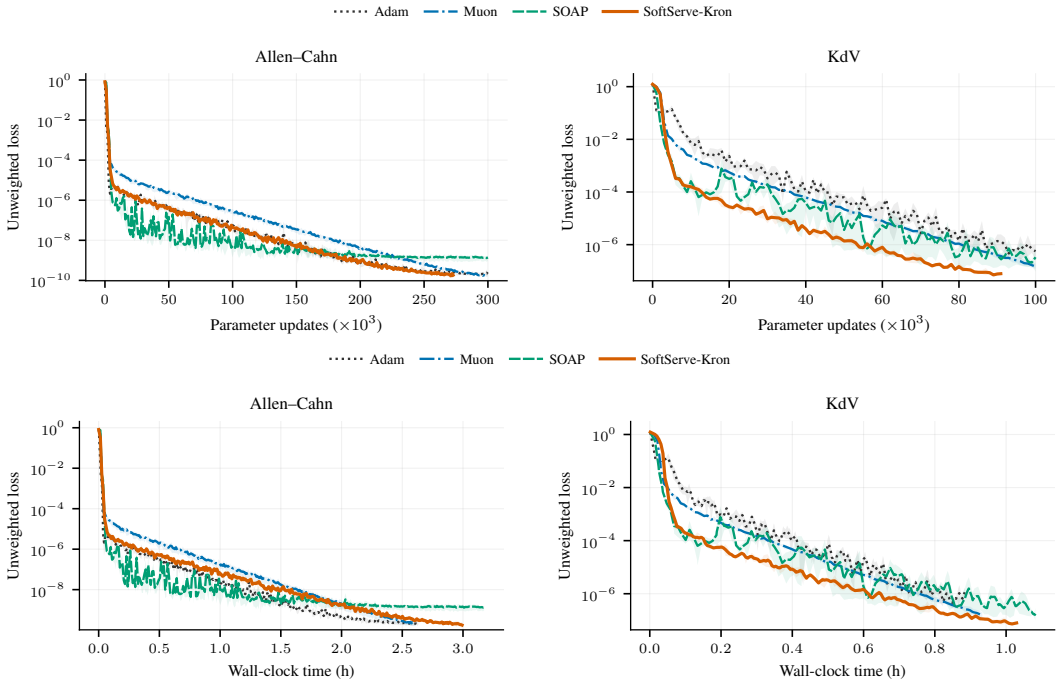


Figure E.14: PirateNet unweighted PDE and initial-condition MSE against updates and wall-clock time. All curves stop at 300,600 gradient evaluations for Allen–Cahn and 100,200 for KdV; Kron therefore uses fewer parameter updates.

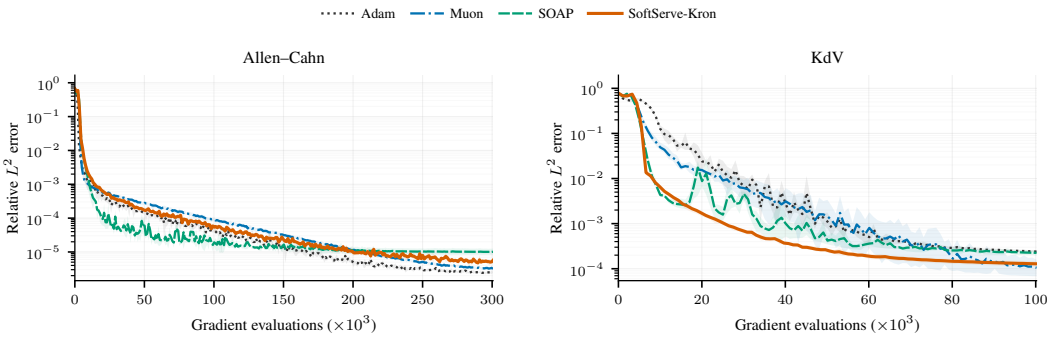


Figure E.15: PirateNet relative L^2 error over the reference space–time grid, up to the same gradient budgets as the training-loss plots. This metric is not used to select the learning rate.

E.6 Physics-informed diffusion

Task and data. We consider structural design using the mechanics dataset of Bastek et al. (2025), which contains 30,000 optimized structures. Given supports, applied loads, and a material budget, the model generates a material-density field and two displacement fields. We retain the released 65×65 field representation and 64×64 finite-element grid.

Model and objective. We use the released 136M-parameter U-Net, consisting of residual convolution blocks and attention layers. It predicts the two displacement fields and the material density, with a sigmoid output for density.

Let z_0 contain the target fields and c the design conditions. At diffusion timestep t, we construct a noisy input

$$z_t = \sqrt{\bar{\alpha}_t} z_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I),$$

where $\bar{\alpha}_t$ is the cumulative signal-retention factor. The network predicts the clean fields as $\widehat{z}_0 = f_\theta(z_t, t, c)$. The training objective combines reconstruction and physics-based terms:

$$\mathcal{L}(\theta) = \mathbb{E}\big[w_t \ \mathrm{MSE}(\widehat{z}_0, z_0) + \ell_t(r_{\mathrm{eq}}) + 10\,\ell_t(r_{\mathrm{vol}}) + 10^{-4} C\big]\,. \tag{E.3}$$

The expectation is over training examples, diffusion timesteps, and noise. The equilibrium residual is

$$r_{\mathrm{eq}} = K(\widehat{\rho})\widehat{u} - f,$$

after applying boundary conditions, where $K(\widehat{\rho})$ is the stiffness matrix determined by the predicted density, $\widehat{u}$ is the predicted displacement, and f is the applied load. The volume residual r_{vol} measures deviation from the material budget. Compliance C measures structural flexibility under the load. We retain the released denoising weights w_t and Gaussian residual penalties ℓ_t, including their diffusion-dependent variances, for all optimizers.

Training and tuning. We use batches of 16, 100 diffusion timesteps, constant learning rates, gradient clipping at 1, and no weight decay. An exponential moving average (EMA) of the parameters uses decay 0.99 after the first 1,000 updates.

Only learning rates are tuned. Each method initially tests five rates for 10,000 updates, extending the grid when needed. Selection uses the EMA validation objective on 128 fixed examples with fixed noise (Figure E.16). Final runs use a budget of 100,000 gradient evaluations.

Momentum and λ remain fixed. Muon uses $\beta_1 = 0.95$ with Nesterov; Adam, SOAP, and SoftServe-Kron use $\beta_1 = 0.9$. Adam and SOAP use $\beta_2 = 0.999$. SoftServe-Kron uses $\lambda = 9$ without Nesterov. Matrix methods use the same convolution-kernel partitions, with blocks of at most 256×256, and Adam at learning rate 10^{-4} on the remaining parameters.

Evaluation. We evaluate the training objective on 32 fixed examples with fixed noise using the current, unaveraged parameters. Validation uses the EMA parameters on 128 separate fixed examples. Figure E.17 shows both objectives against gradient evaluations, parameter updates, and wall-clock time. All curves stop at 100,000 gradient evaluations. Including secant replay, this permits approximately 90,900 parameter updates for SoftServe-Kron and 100,000 for the baselines.

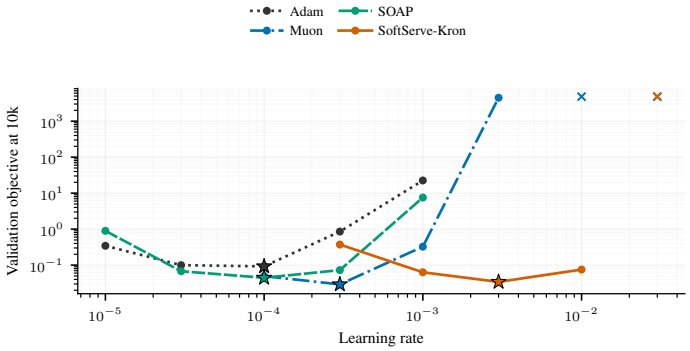


Figure E.16: PIDM LR selection after 10,000 updates. The score is the EMA validation objective. Stars mark selected rates; crosses mark numerical failures.

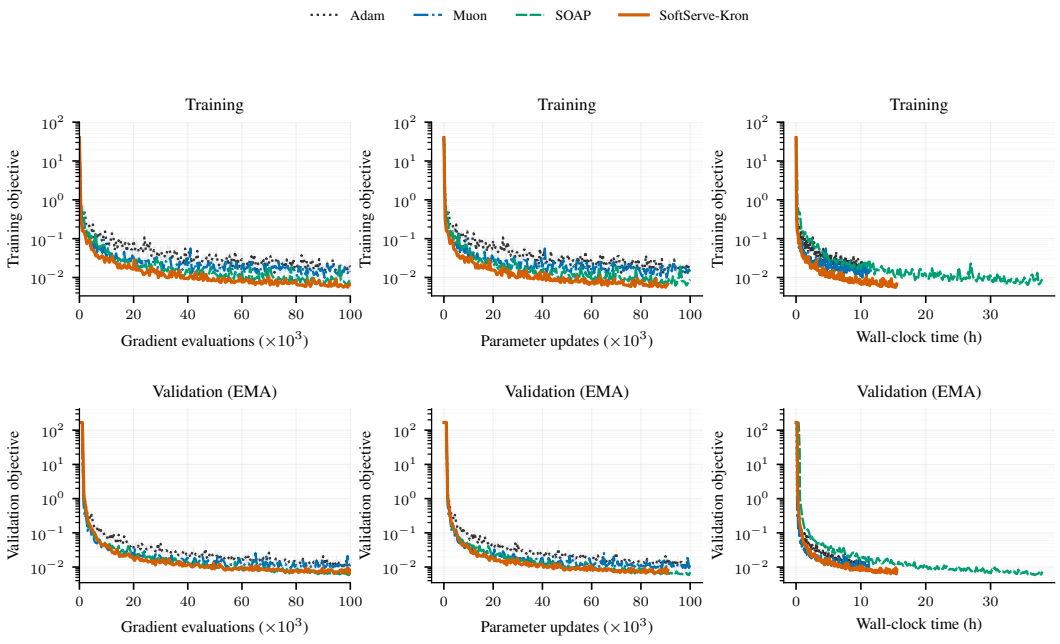


Figure E.17: PIDM training objective (unaveraged parameters) and validation objective (EMA parameters) against gradient evaluations, updates, and wall-clock time. Every curve stops at 100,000 gradient evaluations, corresponding to approximately 90,900 parameter updates for Kron and 100,000 for the baselines.

E.7 Language model

Task and data. We train a language model on 200M tokens from FineWeb (Penedo et al., 2024). The task is to predict each token from its preceding context.

Model and objective. We use a 16.17M-parameter GPT-style model (Radford et al., 2019) with four layers, four attention heads, hidden width 256, and context length 512. We minimize next-token cross-entropy, reported as negative log-likelihood (NLL) per token.

Training and tuning. Each update uses 64 sequences, giving 32,768 tokens per full update. The 200M-token budget requires 6,104 updates, with a smaller final batch. Kron additionally uses 610 same-batch gradient replays. The learning rate warms up over the first 2% of training, remains constant, then decays by cosine to one tenth of its peak over the final 10%. We use no dropout, weight decay, or gradient clipping.

We tune learning rates on seed 0 using five-point sweeps at the full 200M-token budget, selecting by validation NLL. We first tune AdamW, then use its selected peak learning rate of 0.003 for the AdamW updates outside the matrix methods. Muon, SOAP, and Kron update the same attention and MLP weight matrices, with AdamW on the remaining parameters. The selected peak learning rates are 0.003 for Muon and SOAP, and 0.01 for Kron.

Kron uses $\lambda = 99$, $\beta_1 = 0.99$, $K = 10$, and no Nesterov. Muon uses $\beta_1 = 0.9$ with Nesterov. SOAP uses betas $(0.95, 0.95)$ and refreshes its preconditioner every 10 updates. AdamW uses betas $(0.9, 0.95)$, both as a baseline and for the remaining parameters of the matrix methods. Momentum, Nesterov, and λ are fixed throughout tuning.

Evaluation. We evaluate the selected configurations on fresh seeds 1–3 and report their mean NLL. Final training NLL uses one million training tokens; learning-rate selection and final validation NLL use five million held-out tokens.

F Ablation studies

We study the secant penalty λ, the accuracy of the Gram QME solver, and the refresh interval K for SoftServe-Kron. Training studies use RNN Adding and minibatch MNIST, with budgets of 40,000 and 6,000 gradient evaluations, respectively, including secant replay. For each setting, we select the learning rate on seed 0 and evaluate on fresh seeds 1–3. RNN selection uses validation MSE; MNIST selection uses the mean of the last five training objectives. Test errors are not used for selection. Matching completed runs are reused.

Unless varied, $\lambda = 999$, $K = 10$, and the solver uses 18 coupled root iterations and 10 inverse iterations. RNN uses $\beta_1 = 0.99$ and MNIST uses $\beta_1 = 0.95$, both with Nesterov momentum. Lines and points show arithmetic means; bands and error bars span the minimum and maximum over the three seeds.

F.1 Sensitivity to the secant penalty

We vary λ from 0.01 to 9999 on RNN Adding, retuning the learning rate for each value. Figure F.1 shows lower mean validation error as λ increases from 0.01 to 999. Values from 0.1 to 999 outperform the tuned Muon baseline in this experiment. Increasing λ further to 9999 causes numerical failure on all three evaluation seeds. Thus, the penalty can substantially affect both accuracy and stability; the best value here is not a task-independent default.

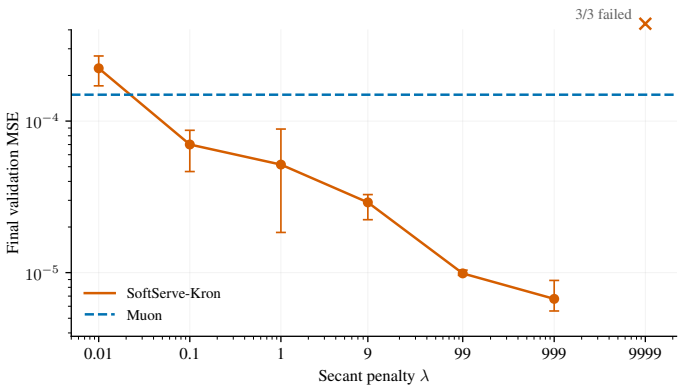


Figure F.1: RNN Adding validation MSE after 40,000 gradient evaluations, with a separately selected learning rate for each λ. Points show means and bars show seed ranges. The dashed line is the mean of the completed Muon comparison. The cross above the axis denotes three numerical failures at $\lambda = 9999$, not a measured loss.

F.2 Accuracy of the QME solver

We first isolate numerical accuracy by solving $\boldsymbol{QUQ} + \boldsymbol{Q} = \boldsymbol{V}$ for fixed 256×256 matrices. We construct $\boldsymbol{U} = \boldsymbol{RR}^\top$ with singular values of $\boldsymbol{R}$ between 0.7 and 1.3, and use independent random eigenbases for $\boldsymbol{U}$ and $\boldsymbol{V}$. The eigenvalues of $\boldsymbol{V}$ are logarithmically spaced from 1 to $\kappa \in \{10, 10^3, 10^5\}$. We vary both iteration counts in the FP32 Gram solver and compare with the FP64 decomposition-based solution $\boldsymbol{Q}_\star$ using relative Frobenius error $\|\widehat{\boldsymbol{Q}} - \boldsymbol{Q}_\star\|_{\mathrm{F}} / \|\boldsymbol{Q}_\star\|_{\mathrm{F}}$.

Figure F.2 shows that too few iterations can leave substantial error. At the default 18/10 counts, median errors are approximately 2.3×10^{-6}, 7.3×10^{-5}, and 7.2×10^{-3} for the three values of κ. More iterations do not uniformly reduce FP32 error, particularly once rounding error limits accuracy.

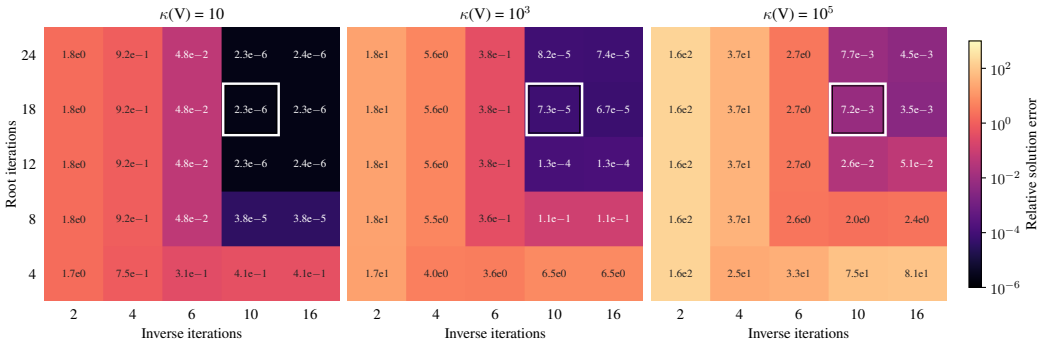


Figure F.2: Relative QME solution error as the root and inverse iteration counts vary. Each cell is the median over three matrix seeds; all panels share a logarithmic color scale. Lower values are better. White outlines mark the default 18 root and 10 inverse iterations.

We then vary the solver counts during RNN training (Figure F.3). We compare 12, 18, and 24 root iterations at 10 inverse iterations, and 10, 16, and 24 inverse iterations at 18 root iterations. Separate sweeps select a learning rate of 1.33×10^{-3} for all four stable settings. Their learning curves are similar; increasing either count beyond 18/10 does not consistently improve validation error. With 12/10, all three tested seed-0 learning rates fail before the full budget, so no fresh-seed mean is shown.

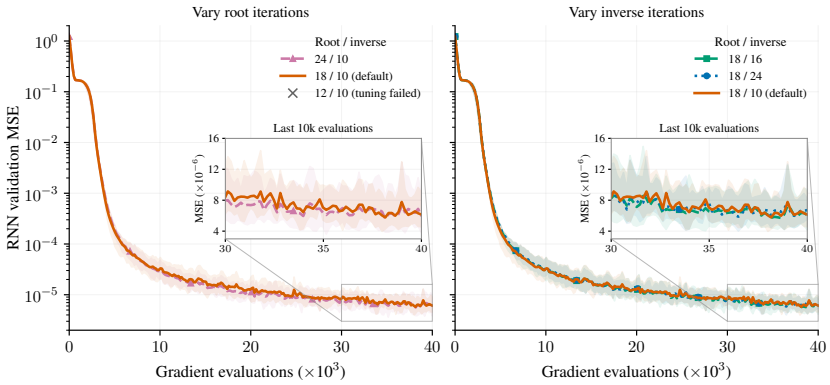


Figure F.3: RNN validation MSE as root iterations vary (left) and inverse iterations vary (right). Legend entries give root/inverse counts; 18/10 is the default. Each completed setting uses a separately selected learning rate and three fresh seeds. Lines show means and shading spans the seed minimum and maximum. Insets enlarge the last 10,000 gradient evaluations on a linear MSE axis with shared limits. The 12/10 setting failed at all three tested tuning rates and has no learning curve.

F.3 Refresh frequency and training cost

We compare $K \in \{1, 5, 10, 20, 50\}$, retuning the learning rate at each interval. Figure F.4 shows learning curves against gradient evaluations and wall-clock time. More frequent refreshes require more secant replays and leave fewer parameter updates within the same budget. Refreshing at every update slows convergence on both tasks. The best trade-off is task-dependent: $K = 5$ reaches the lowest mean RNN validation MSE, while larger intervals shorten MNIST training with similar final objectives.

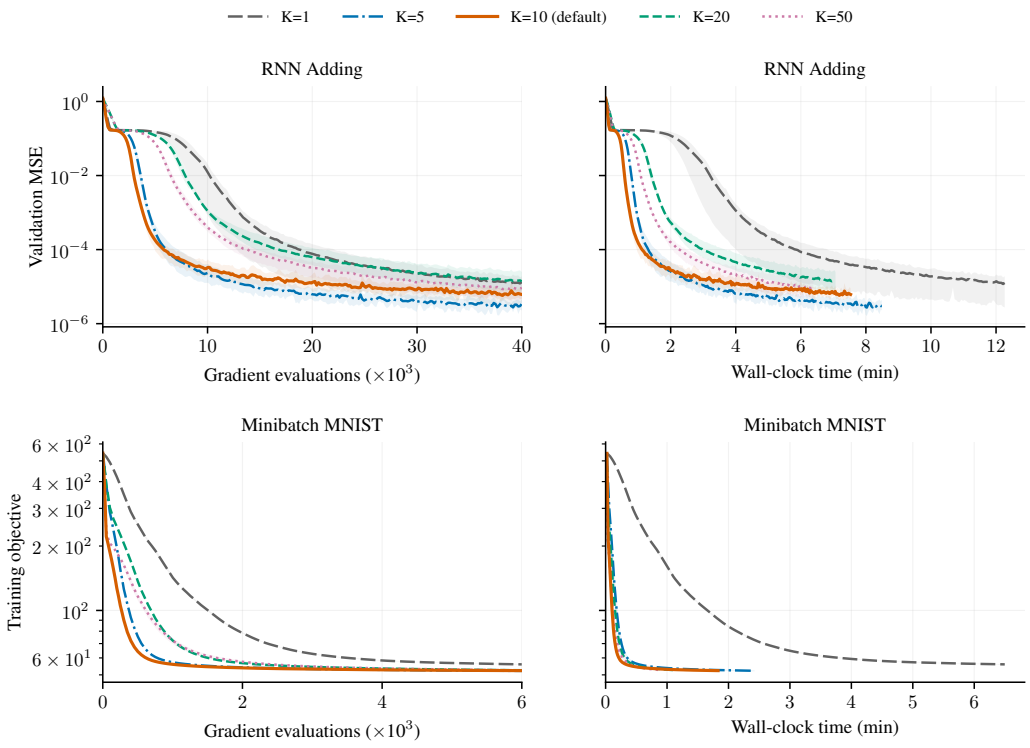


Figure F.4: Learning curves for different refresh intervals. Rows show RNN validation MSE and minibatch MNIST training objective; columns show gradient evaluations and wall-clock time. All runs stop at the same task-specific gradient budget, including replay: 40,000 for RNN and 6,000 for MNIST. Lines show three-seed means and shading spans the seed minimum and maximum. Wall-clock time includes evaluation and diagnostic overhead. Orange lines mark the default $K = 10$.