MINN: A MATH-INFORMED NEURAL OPTIMIZER WITH GLOBAL FIRST-ORDER CONVERGENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

We present **MINN** (**M**ath-**I**nformed **N**eural **N**etwork optimizer), a hybrid learned–classical optimizer for unconstrained nonlinear optimization. Purely learned optimizers can provide fast amortized progress, but often stall or fail under distribution shift; classical methods are more reliable, but may be slower and less adaptive. MINN bridges this gap through a principled two-stage architecture that separates neural *proposal* from classical *certification*. At each outer iteration, a coordinate-wise LSTM direction is combined with a steepest-descent direction through a bounded scalar trust mechanism. A sufficient-descent projection safeguard and Armijo backtracking line search then produce a safeguarded Stage-1 step with sufficient decrease. The safeguarded point is further refined by an inner loop of learnable L-BFGS-style steps, where memory-slot weights and step sizes are trainable. An accept/reject monotonicity filter accepts this refinement only if it does not increase the objective relative to the safeguarded point, allowing the full method to inherit monotone decrease and *global first-order convergence* under the stated assumptions on the objective and safeguarded updates. All parameters, including the LSTM weights, trust-head parameters, L-BFGS memory weights, and step-size parameters, are trained end-to-end using an unsupervised amortized loss with an increasing-horizon curriculum. Experiments on least-squares, neural network training, and nonconvex Rastrigin benchmarks show that MINN improves the speed–accuracy trade-off, generalizes across dimensions and distributions, and avoids the stalling and out-of-distribution failures of purely learned optimizers.

1 INTRODUCTION

Nonlinear optimization is a fundamental primitive in machine learning, inverse problems, and scientific computing. In many applications, however, one does not solve a single problem once; one repeatedly solves a *distribution* of related instances. This repeated-structure regime naturally motivates *amortized optimization*: if an optimizer can learn reusable update patterns from past instances, it may reduce test-time computation on new ones (Amos, 2025).

This motivation underlies the growing literature on learned optimizers, especially recurrent and coordinate-wise architectures that directly predict iterative updates (Andrychowicz et al., 2016; Lv et al., 2017; Chen et al., 2022). Such methods can be strikingly effective in the early phase of optimization, where amortized priors help them make rapid progress. Yet their practical reliability remains limited. Purely learned optimizers typically provide no per-instance descent guarantee, often stall before reaching high accuracy, and can generalize poorly under changes in dimension, conditioning, or problem distribution (Gao et al., 2024; Chen et al., 2022). In contrast, classical methods such as gradient descent and L-BFGS are slower to exploit cross-instance structure, but they remain robust because they are built around principled mechanisms for descent, curvature use, and globalization (Nocedal and Wright, 2006).

This tension suggests a natural goal: combine the *adaptivity* of learned optimizers with the *stability* of classical optimization. Achieving this is subtle. Injecting a neural update into a classical solver can destroy descent, make line search ineffective, or allow an aggressive learned refinement to undo objective decrease. A useful hybrid optimizer must therefore do more than append a neural module to a classical routine: it must embed learning *inside* a mathematically controlled structure.

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.

We propose **MINN** (**M**ath-**I**nformed **N**eural **N**etwork optimizer), a two-stage solver for unconstrained nonlinear optimization. Stage 1 combines a coordinate-wise LSTM proposal with the negative gradient through bounded trust fusion, then certifies a step by sufficient-descent projection and Armijo backtracking. Stage 2 applies weighted L-BFGS and keeps its endpoint only if it preserves the Stage-1 decrease. The neural trial trajectory supplies both the proposal and curvature pairs that initialize the refinement's memory, reusing gradients already evaluated along the trial path. Stage 2 evaluates gradients at its own iterates.

The LSTM, trust head, and learnable L-BFGS parameters are trained jointly with an unsupervised amortized objective. Their shared parameterization allows use across dimensions. Learning proposes updates, while classical safeguards enforce sufficient decrease and first-order convergence under the stated assumptions.

Contributions. Our main contributions are:

(1) We propose MINN, a novel two-stage hybrid optimizer that combines a coordinate-wise LSTM with a learnable L-BFGS method through a bounded scalar trust mechanism, sufficient-descent projection, and monotonicity filter.

(2) We prove that the full two-stage algorithm achieves global first-order convergence under the stated assumptions. The normalized bounded fusion and fixed safeguards enforce the needed conditions without assuming that the LSTM proposal is a descent direction.

(3) We demonstrate empirically that MINN achieves state-of-the-art speed among learned optimizers while maintaining the robustness and solution quality of classical methods across in-distribution, out-of-distribution, and cross-dimensional evaluation protocols.

2 Related Work

Learning to Optimize. Learned optimizers replace or augment hand-designed update rules. Andrychowicz et al. (2016) meta-learn coordinate-wise LSTM updates; Wichrowska et al. (2017) use a multi-timescale RNN that transfers across problem sizes. Lv et al. (2017) improve generalization and long-horizon behavior with Adam-like gradient preprocessing and random scaling during training. Symbolic discovery instead searches over update rules (Chen et al., 2023).

Hybrid Neural–Classical Optimization. Algorithm unrolling maps classical iterations to network layers, as in LISTA's learned ISTA parameters (Gregor and LeCun, 2010); Monga et al. (2021) survey this approach. Other hybrids learn proximal operators (Meinhardt et al., 2017), step-size policies (Daniel et al., 2016), feasibility-seeking updates (Nguyen and Donti, 2025), or preconditioners (Häusner et al., 2023), retaining classical solver structure while learning selected components.

Safeguarded and convergence-guaranteed L2O. Examples of wrapping a learned update in a test that restores classical guarantees include Heaton et al. (2023), who accept an L2O step in a convex fixed-point setting only when it reduces a fixed-point residual below a decaying tolerance, and otherwise substitute the classical operator step; and Prémont-Schwarz et al. (2022), whose loss-guarded L2O compares the learned optimizer's objective value against a reference SGD step and switches optimizers when the learned one is worse. A second line restricts trainable solver parameters to a convergent region (Banert et al., 2020) or selects learned updates from a set on which convergence is proved (Banert et al., 2024). Related work derives mathematical structures as necessary conditions for convergence rather than sufficient guarantees (Liu et al., 2023). A third line learns objects inside a classical scheme — quasi-Newton metrics (Liao et al., 2023), preconditioners (Häusner et al., 2023), step sizes (Daniel et al., 2016). Safeguarded Anderson acceleration similarly combines accelerated steps with a test on the fixed-point residual and an averaged fixed-point fallback (Zhang et al., 2020). MINN belongs to this safeguarded family. Descent-half-space projection and line search also appear in Moeller et al. (2019); Appendix K details the comparison.

3 THE MINN HYBRID ALGORITHM

We consider a family of smooth unconstrained optimization problems

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x};\xi), \tag{1}$$

where $\boldsymbol{x}$ is the decision variable and $\xi \in \mathcal{C} \subseteq \mathbb{R}^{d_\xi}$ specifies the instance (e.g., regression data or physical-system parameters). The objective is $f : \mathbb{R}^n \times \mathcal{C} \to \mathbb{R}$. When a minimizer exists, $\boldsymbol{x}^\star(\xi)$ denotes one selected minimizer.

In amortized optimization, instances are sampled from a distribution $\mathcal{D}$, and training seeks reusable information that reduces the cost of subsequent solves. A direct model $\widehat{\boldsymbol{x}}_\vartheta : \mathcal{C} \to \mathbb{R}^n$ predicts $\widehat{\boldsymbol{x}}_\vartheta(\xi) \approx \boldsymbol{x}^\star(\xi)$. MINN instead learns an iterative update rule, continuing to evaluate the objective and gradient for each new instance. We suppress ξ when clear, use Euclidean norms, and differentiate f with respect to $\boldsymbol{x}$.

Certify progress, then refine it. At outer iterate $\boldsymbol{x}_t$, Stage 1 runs an LSTM along a trial path, blends its proposed direction with $-\boldsymbol{g}_t$, where $\boldsymbol{g}_t = \nabla f(\boldsymbol{x}_t)$, and certifies a step by projection and Armijo search. Stage 2 starts at the certified point $\hat{\boldsymbol{x}}_t$, uses trial-path curvature to initialize a short learnable L-BFGS run, and returns an endpoint $\boldsymbol{z}_t$. The next iterate is $\boldsymbol{z}_t$ if $f(\boldsymbol{z}_t) \le f(\hat{\boldsymbol{x}}_t)$, and $\hat{\boldsymbol{x}}_t$ otherwise (Figure 1 and Algorithm 1).

The learned parameters are $\Theta = (\theta, \phi, \psi)$: θ contains the shared LSTM and direction-readout parameters (Appendix C); ϕ contains the scalar trust-head parameters (Section 3.2); and $\psi = \{\boldsymbol{v}, \tilde{\boldsymbol{\mu}}\}$ contains the L-BFGS memory-slot logits and inner-step parameters (Section 3.4). Projection, line search, memory filtering, and endpoint acceptance have no trainable parameters. The indices t, h, k, i, j denote the outer iteration, LSTM trial step, refinement step, memory slot, and coordinate, respectively; Appendix A collects the notation.

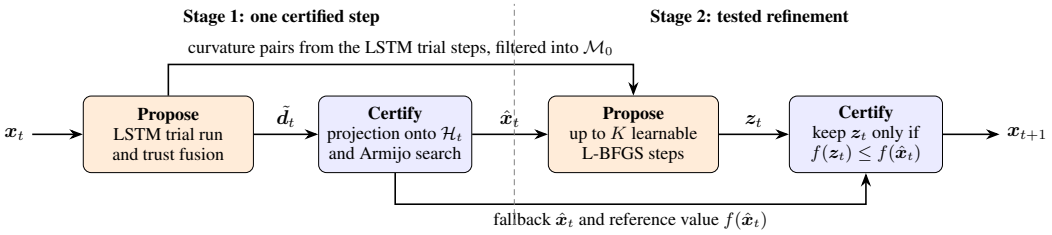

Figure 1: MINN's two-stage interface. Learned components (orange) propose; classical tests (blue) certify. The trial run supplies both a direction and curvature pairs. The certified point $\hat{\boldsymbol{x}}_t$ starts Stage 2 and remains the fallback if refinement fails its endpoint test.

Algorithm 1 One MINN outer iteration. Algorithm 4 additionally checks the gradient tolerance before each stage; a Stage-2 skip returns $\hat{\boldsymbol{x}}_t$.

Require: $\boldsymbol{x}_t$, $\boldsymbol{g}_t = \nabla f(\boldsymbol{x}_t) \neq \boldsymbol{0}$, carried LSTM states
Ensure: $\boldsymbol{x}_{t+1}$ with $f(\boldsymbol{x}_{t+1}) \le f(\hat{\boldsymbol{x}}_t)$

1:	*Stage 1: one certified step*	
2:	Run H LSTM trial steps; retain $\boldsymbol{d}_t^{\mathrm{lstm}}$, $\bar{\boldsymbol{h}}_t$, and the last m available curvature pairs	(2)–(4)
3:	Compute trust weights and form the fused proposal $\tilde{\boldsymbol{d}}_t$	(5)–(6)
4:	Project $\tilde{\boldsymbol{d}}_t$ onto $\mathcal{H}_t$ to obtain $\boldsymbol{d}_t$	(7)–(8)
5:	Armijo backtracking gives $\hat{\boldsymbol{x}}_t = \boldsymbol{x}_t + \eta_t \boldsymbol{d}_t$	(9)–(10)
6:	*Stage 2: tested refinement*	
7:	Filter trial pairs into $\mathcal{M}_0$; run up to K learnable L-BFGS steps from $\hat{\boldsymbol{x}}_t$ to get $\boldsymbol{z}_t$	Algorithm 2
8:	Set $\boldsymbol{x}_{t+1} = \boldsymbol{z}_t$ if $f(\boldsymbol{z}_t) \le f(\hat{\boldsymbol{x}}_t)$; otherwise set $\boldsymbol{x}_{t+1} = \hat{\boldsymbol{x}}_t$	(13)

3.1 THE LSTM PROPOSAL $\mathcal{F}_\theta$

Following Andrychowicz et al. (2016), one LSTM cell with parameters θ is shared across coordinates and trial steps. Starting at $\boldsymbol{x}_{t,0}^{\mathrm{lstm}} = \boldsymbol{x}_t$, step $h = 0, \ldots, H-1$ processes coordinate values and

gradients $\boldsymbol{g}_{t,h}^{\mathrm{lstm}} = \nabla f(\boldsymbol{x}_{t,h}^{\mathrm{lstm}})$ and outputs an increment $\boldsymbol{d}_{t,h}^{\mathrm{lstm}}$, giving $\boldsymbol{x}_{t,h+1}^{\mathrm{lstm}} = \boldsymbol{x}_{t,h}^{\mathrm{lstm}} + \boldsymbol{d}_{t,h}^{\mathrm{lstm}}$. Each coordinate has its own hidden state $\boldsymbol{h}_{t,h}^{(j)} \in \mathbb{R}^D$ and cell state. These states carry across outer iterations, while the trial path restarts at the current iterate; cell equations and the readout are in Appendix C.

The trial run returns three quantities. First, its last increment is the candidate direction

$$\boldsymbol{d}_t^{\mathrm{lstm}} = \big(d_{t,H-1}^{\mathrm{lstm},(j)}\big)_{j=1}^{n}. \tag{2}$$

This uses all H processed gradients, but is computed at $\boldsymbol{x}_{t,H-1}^{\mathrm{lstm}}$, not generally at $\boldsymbol{x}_t$. It is neither the total trial displacement nor a certified descent direction at $\boldsymbol{x}_t$. The trial points are therefore not accepted directly. Second, for $1 \le m \le H-1$, the last m segments with evaluated endpoint gradients yield

$$\boldsymbol{s}_h^{\mathrm{lstm}} = \boldsymbol{x}_{t,h}^{\mathrm{lstm}} - \boldsymbol{x}_{t,h-1}^{\mathrm{lstm}}, \qquad \boldsymbol{y}_h^{\mathrm{lstm}} = \boldsymbol{g}_{t,h}^{\mathrm{lstm}} - \boldsymbol{g}_{t,h-1}^{\mathrm{lstm}}, \qquad h = H-m, \dots, H-1. \tag{3}$$

These displacement–gradient-change pairs capture trial-path curvature without extra gradient evaluations; the index t is suppressed. Third, the pooled final hidden state

$$\bar{\boldsymbol{h}}_t = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{h}_{t,H-1}^{(j)} \in \mathbb{R}^D \tag{4}$$

summarizes the trial run for the trust head. The notation $\mathcal{F}_\theta(\boldsymbol{x}_t, H)$ denotes these outputs, with dependence on the carried states implicit.

3.2 Bounded Scalar Trust Fusion

The trust head balances the learned proposal against the current negative gradient using

$$\tau_t = \tau_{\min} + (\tau_{\max} - \tau_{\min})\sigma(\boldsymbol{w}_\tau^\top \bar{\boldsymbol{h}}_t + b_\tau), \qquad \nu_t = \nu_{\max}\sigma(\boldsymbol{w}_\nu^\top \bar{\boldsymbol{h}}_t + b_\nu), \tag{5}$$

where σ is logistic, $0 < \tau_{\min} < \tau_{\max}$, $\nu_{\max} > 0$, and $\phi = \{\boldsymbol{w}_\tau, b_\tau, \boldsymbol{w}_\nu, b_\nu\}$. MINN forms

$$\tilde{\boldsymbol{d}}_t = -\tau_t \boldsymbol{g}_t + \nu_t \boldsymbol{d}_t^{\mathrm{lstm,n}}, \qquad \boldsymbol{d}_t^{\mathrm{lstm,n}} = \frac{\|\boldsymbol{g}_t\|}{\|\boldsymbol{d}_t^{\mathrm{lstm}}\| + \varepsilon_{\mathrm{dir}}} \boldsymbol{d}_t^{\mathrm{lstm}}, \tag{6}$$

with $\varepsilon_{\mathrm{dir}} > 0$. Normalization gives $\|\boldsymbol{d}_t^{\mathrm{lstm,n}}\| \le \|\boldsymbol{g}_t\|$, making the weights comparable and ensuring $\|\tilde{\boldsymbol{d}}_t\| \le (\tau_{\max} + \nu_{\max})\|\boldsymbol{g}_t\|$. An uphill neural contribution can nevertheless overwhelm $-\tau_t \boldsymbol{g}_t$, so bounded fusion alone does not certify descent.

3.3 Stage 1: Projection and Armijo Certification

For $\kappa > 0$, sufficient descent requires membership in

$$\mathcal{H}_t = \{\boldsymbol{d} \in \mathbb{R}^n : \boldsymbol{g}_t^\top \boldsymbol{d} \le -\kappa \|\boldsymbol{g}_t\|^2\}. \tag{7}$$

MINN makes the least Euclidean correction to the fused proposal:

$$\boxed{\boldsymbol{d}_t = \tilde{\boldsymbol{d}}_t - \frac{\max\{0, \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa\|\boldsymbol{g}_t\|^2\}}{\|\boldsymbol{g}_t\|^2}\boldsymbol{g}_t.} \tag{8}$$

MINN stops before this operation if $\|\boldsymbol{g}_t\| < \varepsilon_{\mathrm{tol}}$, so $\boldsymbol{g}_t \neq \boldsymbol{0}$. Projection is inactive for a proposal already in $\mathcal{H}_t$; otherwise it changes only the component along $\boldsymbol{g}_t$. It also preserves a bound proportional to $\|\boldsymbol{g}_t\|$, possibly with a larger constant.

Why project the combined direction? Descent is a property of the sum: the neural contribution can cancel part of the negative-gradient contribution while leaving sufficient descent. Projecting only the neural term onto $\{\boldsymbol{u} : \boldsymbol{g}_t^\top \boldsymbol{u} \le 0\}$ also works when $\tau_t \ge \kappa$, but may remove more of the proposal. Projecting the sum makes the smallest correction and also handles $\tau_t < \kappa$.

Lemma 1 (Descent guarantee)**.** *If $\boldsymbol{x}_t$ is not a stationary point, then $\boldsymbol{d}_t$ is a sufficient descent direction.*

A descent direction may still give an overly long step. Armijo backtracking chooses the largest $\eta_t \in \{1, \beta, \beta^2, \dots\}$, $\beta \in (0,1)$, satisfying

$$f(\boldsymbol{x}_t + \eta_t \boldsymbol{d}_t) \le f(\boldsymbol{x}_t) + c_1 \eta_t \boldsymbol{g}_t^\top \boldsymbol{d}_t, \qquad c_1 \in (0,1), \tag{9}$$

and returns the certified point

$$\hat{\boldsymbol{x}}_t = \boldsymbol{x}_t + \eta_t \boldsymbol{d}_t. \tag{10}$$

3.4 Stage 2: Learnable L-BFGS Refinement $\mathcal{S}_\psi$

Stage 2 receives $\hat{\boldsymbol{x}}_t$ and the initial curvature memory, not the LSTM state. Its points start at $\boldsymbol{z}_{t,0} = \hat{\boldsymbol{x}}_t$. At step k, $\boldsymbol{g}_{t,k} = \nabla f(\boldsymbol{z}_{t,k})$ gives the direction $\boldsymbol{p}_k = -\boldsymbol{W}_k \boldsymbol{g}_{t,k}$, where a two-loop recursion applies the inverse-Hessian approximation $\boldsymbol{W}_k$ without forming it. The trial pairs reuse gradients already evaluated to seed the curvature model; new pairs adapt it during refinement. Learning controls the strength of the curvature updates through memory-slot weights and the distance moved through an inner-step schedule. These parameters are trained offline, not predicted from the current pair values; the operator and search direction still adapt through measured curvature and the current gradient. The usefulness of trial-path curvature at $\hat{\boldsymbol{x}}_t$ is an empirical design choice, while the safeguards control the accepted progress.

Memory and learned weights. The seed memory $\mathcal{M}_0$ contains the pairs (3) satisfying $\boldsymbol{y}^\top \boldsymbol{s} > \varepsilon_{\text{curv}} > 0$, oldest first. The certified displacement $\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t$ is not added as a pair. Before step k, let $\mathcal{M}_k = \{(\boldsymbol{s}_i, \boldsymbol{y}_i)\}_{i=1}^{m_k}$, $m_k \le m$. For $m_k > 0$, MINN replaces the classical coefficient $1/(\boldsymbol{y}_i^\top \boldsymbol{s}_i)$ by

$$\boldsymbol{\pi} = \operatorname{softmax}(\boldsymbol{v}_{1:m_k}), \qquad \rho_i = \frac{\pi_i}{\boldsymbol{y}_i^\top \boldsymbol{s}_i + \varepsilon_{\text{den}}}, \qquad i = 1, \dots, m_k. \tag{11}$$

Here $\varepsilon_{\text{den}} > 0$. The logits attach to memory positions and are fixed after training; normalized weights depend on m_k, not on the pair values. For $m_k \ge 2$, each $\pi_i < 1$, so no ρ_i equals its classical value. The initial scaling and operator derivation are in Appendix D.1. Positive-curvature filtering makes $\boldsymbol{W}_k$ positive definite, hence $\boldsymbol{p}_k$ a descent direction when $\boldsymbol{g}_{t,k} \neq \boldsymbol{0}$; empty memory gives $\boldsymbol{W}_k = \boldsymbol{I}$.

Steps and output. The update is

$$\boldsymbol{z}_{t,k+1} = \boldsymbol{z}_{t,k} + \mu_k \boldsymbol{p}_k, \qquad \mu_k = \operatorname{softplus}(\tilde{\mu}_k) > 0. \tag{12}$$

After each step, $(\boldsymbol{z}_{t,k+1} - \boldsymbol{z}_{t,k}, \boldsymbol{g}_{t,k+1} - \boldsymbol{g}_{t,k})$ enters the memory if it passes the same curvature test; the oldest pair is removed when capacity m is exceeded. The m logits and K step parameters $\psi = \{\boldsymbol{v}, \tilde{\boldsymbol{\mu}}\}$ are shared across instances and outer iterations. Training initializes each μ_k to one; there is no inner line search. The run ends after K steps or when $\|\boldsymbol{g}_{t,k+1}\| < \varepsilon_{\text{tol}}$, returns its last point $\boldsymbol{z}_t = \mathcal{S}_\psi(\hat{\boldsymbol{x}}_t, \mathcal{M}_0)$, and discards the memory.

3.5 Combining the Stages: Endpoint Acceptance

Positive inner step lengths need not decrease f. MINN therefore selects between two existing points:

$$\boxed{\boldsymbol{x}_{t+1} = \begin{cases} \boldsymbol{z}_t, & f(\boldsymbol{z}_t) \le f(\hat{\boldsymbol{x}}_t), \\ \hat{\boldsymbol{x}}_t, & \text{otherwise.} \end{cases}} \tag{13}$$

Testing against $\hat{\boldsymbol{x}}_t$ preserves the full Stage-1 decrease:

$$f(\boldsymbol{x}_{t+1}) \le f(\hat{\boldsymbol{x}}_t) \le f(\boldsymbol{x}_t) - c_1 \kappa \eta_t \|\boldsymbol{g}_t\|^2. \tag{14}$$

This is the only Stage-2 property used by Proposition 2. The next trial run starts at $\boldsymbol{x}_{t+1}$ with the carried LSTM states, whether refinement was accepted or rejected.

3.6 Training Objective

For context $\xi \sim \mathcal{D}$ and initialization $\boldsymbol{x}_0 \sim \mathcal{P}_0(\cdot \mid \xi)$, unroll T outer iterations and minimize

$$\mathcal{L}(\Theta; \xi, \boldsymbol{x}_0) = \sum_{t=1}^{T} \omega_t \big[f(\boldsymbol{x}_t; \xi) + f(\hat{\boldsymbol{x}}_{t-1}; \xi) \big] + \lambda \sum_{t=1}^{T} \omega_t \big[\|\nabla f(\boldsymbol{x}_t; \xi)\|^2 + \|\nabla f(\hat{\boldsymbol{x}}_{t-1}; \xi)\|^2 \big], \tag{15}$$

with $\omega_t > 0$ and $\lambda \ge 0$. Including the certified points trains Stage 1 directly even when refinement is accepted. No minimizer labels are needed for the population objective

$$\boxed{\min_{\Theta} \mathbb{E}_{\substack{\xi \sim \mathcal{D} \\ \boldsymbol{x}_0 \sim \mathcal{P}_0(\cdot \mid \xi)}} [\mathcal{L}(\Theta; \xi, \boldsymbol{x}_0)].} \tag{16}$$

Algorithm 3 uses Adam, an increasing horizon, and truncated backpropagation (Appendix E). Differentiation through objective gradients can require Hessian-vector products where the relevant derivatives exist; branch switches need not be differentiable, and truncation approximates full meta-gradients. At inference the parameters are fixed and no meta-gradients are computed.

4 Convergence Analysis

We now prove convergence properties for inference (see Algorithm 4 in the Appendix). We establish that MINN's normalized bounded fusion and half-space correction satisfy classical sufficient-descent conditions, and that Armijo backtracking and endpoint acceptance preserve global first-order convergence under the stated assumptions. Here first-order convergence concerns stationarity, not attainment of a global minimum.

Proposition 1 (Global convergence of the safeguarded Stage-1 iteration)**.** *Consider the Stage-1 iteration defined as follows. Given $\boldsymbol{x}_t \in \mathbb{R}^n$, if $\boldsymbol{g}_t = \nabla f(\boldsymbol{x}_t) = \mathbf{0}$, terminate. Otherwise, define $\boldsymbol{d}_t$ as in (8) and let η_t be chosen by Armijo backtracking (9). The next iterate is $\boldsymbol{x}_{t+1} = \hat{\boldsymbol{x}}_t = \boldsymbol{x}_t + \eta_t \boldsymbol{d}_t$. Assume:*

- *(A1) $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and bounded below on the level set $\mathcal{X}_0 := \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) \le f(\boldsymbol{x}_0)\}$.*
- *(A2) ∇f is globally Lipschitz continuous on $\mathbb{R}^n$ with Lipschitz constant $L > 0$.*
- *(A3) For all t, the trust coefficients produced by (5) satisfy $0 < \tau_{\min} \le \tau_t \le \tau_{\max}$ and $0 \le \nu_t \le \nu_{\max}$.*

Then either the algorithm terminates finitely at a stationary point, or it generates an infinite sequence $\{\boldsymbol{x}_t\}$ such that

$$\liminf_{t\to\infty} \|\nabla f(\boldsymbol{x}_t)\| = 0. \tag{17}$$

Proposition 2 (Global convergence of the full two-stage method with accept/reject safeguard)**.** *Assume (A1)–(A3) of Proposition 1. At each outer iteration t, proceed as follows:*

- *(i) If $\boldsymbol{g}_t = \nabla f(\boldsymbol{x}_t) = \mathbf{0}$, terminate.*
- *(ii) Otherwise, form the Stage-1 point $\hat{\boldsymbol{x}}_t$ of (10), with $\boldsymbol{d}_t$ from (8) and η_t from (9).*
- *(iii) Let $\boldsymbol{z}_t \in \mathbb{R}^n$ be arbitrary, for example $\boldsymbol{z}_t = \mathcal{S}_\psi(\hat{\boldsymbol{x}}_t, \mathcal{M}_0)$ of Section 3.4.*
- *(iv) Set $\boldsymbol{x}_{t+1}$ by the acceptance test (13).*

Then either the two-stage algorithm terminates finitely at a stationary point, or it generates an infinite sequence $\{\boldsymbol{x}_t\}$ such that

$$\liminf_{t\to\infty} \|\nabla f(\boldsymbol{x}_t)\| = 0. \tag{18}$$

If, in addition, $\mathcal{X}_0$ is compact, then every accumulation point of $\{\boldsymbol{x}_t\}$ is a stationary point of f.

Appendices G.1–G.3 prove Lemma 1 and Propositions 1–2.

5 Experiments

We evaluate MINN on three standard benchmarks from the learning-to-optimize literature: least-squares problems, neural network training on MNIST, and non-convex Rastrigin function minimization (Chen et al., 2022; Andrychowicz et al., 2016). We assess *in-distribution* performance and, for least squares and MNIST, *out-of-distribution* (OOD) generalization. All learned optimizers are trained at a single problem scale and tested at larger dimensions or on modified problem structures without retraining. We use three problem classes:

(i) **Random least-squares.** $\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$, where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ has i.i.d. entries $A_{ij} \sim \mathcal{N}(0,1)$, $\boldsymbol{x}^\star \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_n)$, and $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}^\star$. We generate 20,000 training instances at n=200 and 100 test instances for each $n \in \{200, 400, 600, 800, 1000\}$.

(ii) **Neural network training (MNIST MLP).** Following Andrychowicz et al. (2016), we optimize a $784 \to 20 \to 10$ MLP with sigmoid activation on MNIST (cross-entropy loss). The decision variable $\boldsymbol{x} \in \mathbb{R}^{15{,}910}$ collects all weights and biases. The task distribution consists of different random initializations $\boldsymbol{x}_0 \sim \mathcal{N}(\mathbf{0}, 0.1\,\boldsymbol{I}_n)$.

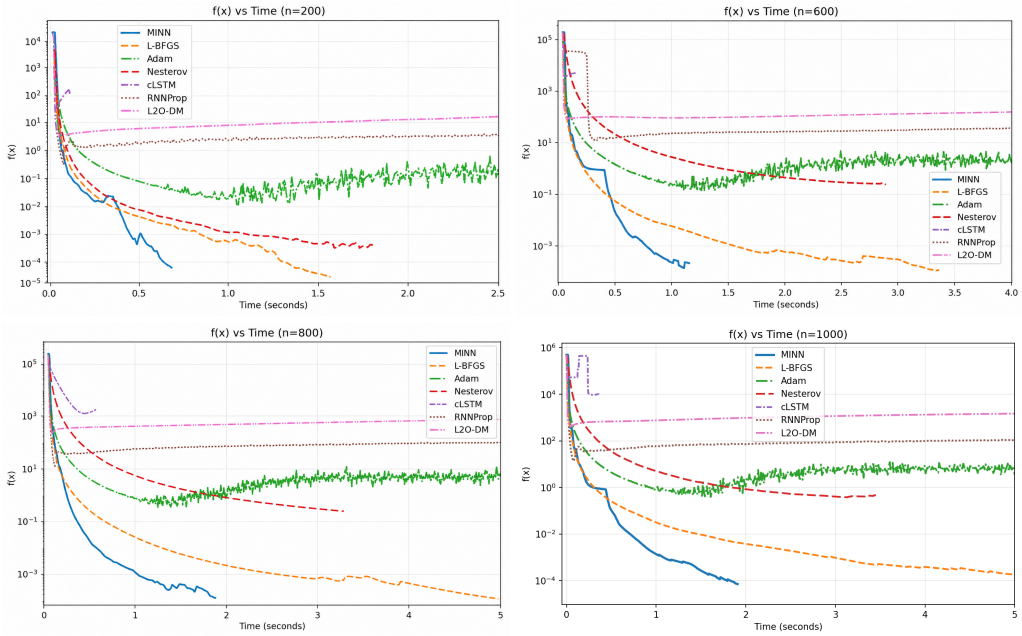

Figure 2: Random least-squares (in-distribution): convergence of $f(\boldsymbol{x}_t)$ vs. wall-clock time.

(iii) **Non-convex Rastrigin.** $f_q(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}_q\boldsymbol{x} - \boldsymbol{b}_q\|_2^2 - \alpha\, \boldsymbol{c}_q^\top \cos(2\pi\boldsymbol{x}) + \alpha n$, where q indexes the problem instance, $\boldsymbol{A}_q \in \mathbb{R}^{n\times n}$, $\boldsymbol{b}_q, \boldsymbol{c}_q \in \mathbb{R}^n$, and the cosine acts component-wise. We use α=0.5 and sample all entries of $\boldsymbol{A}_q$, $\boldsymbol{b}_q$, and $\boldsymbol{c}_q$ independently from $\mathcal{N}(0,1)$. We train on 20,000 instances at n=200 and test on 100 instances per dimension $n \in \{200, 400, 600, 800, 1000\}$; see Figure 6 and Table 6 in the appendix.

Out-of-distribution protocols. For least-squares, we shift the test distribution to $A_{ij} \sim \mathcal{N}(0.15, 1)$. For MNIST, we train on sigmoid activations and test on ReLU activations (the activation-transfer protocol of Andrychowicz et al., 2016).

Baselines. We compare against three classical optimizers—**L-BFGS**, **Adam** (Kingma and Ba, 2015), and **Nesterov accelerated gradient** (Nesterov, 1983)—implemented via PyTorch, and three learned optimizers: the coordinate-wise **cLSTM** (Andrychowicz et al., 2016), **RNNProp** (Lv et al., 2017) (LSTM with learned momentum), and **L2O-DM** (dual-memory LSTM with attention), following the baseline suite of Chen et al. (2022). All learned baselines share parameters across coordinates and are trained on the same data as MINN.

5.1 Experiment 1: Least-Squares

In-distribution. Learned optimizers are trained on least-squares instances with $A_{ij} \sim \mathcal{N}(0, 1)$ at n=200, and all methods are tested on the same distribution for $n \in \{200, 400, 600, 800, 1000\}$. Figure 2 shows convergence of $f(\boldsymbol{x}_t)$ versus wall-clock time for $n \in \{200, 600, 800, 1000\}$.

The trajectories reveal a clear trade-off. Purely learned optimizers (cLSTM, RNNProp, and L2O-DM) make rapid initial progress from amortization, but quickly plateau and fail to reach the high-accuracy regime, consistent with prior observations on learned optimizers (Gao et al., 2024; Chen et al., 2022). Their brittleness becomes severe under scale shift: for $n > 200$, the learned baselines become unstable or diverge. Classical methods show the opposite behavior: L-BFGS, Adam, and Nesterov are slower initially but continue improving after the learned baselines stall, with L-BFGS and Nesterov reaching much higher final accuracy. MINN combines both regimes. It decreases f as quickly as learned methods early on, but continues to improve like a classical method. Across all tested dimensions, MINN remains stable and reaches low objective values quickly, suggesting that the coordinate-wise LSTM provides amortized, dimension-agnostic directions while the learnable L-BFGS stage supplies

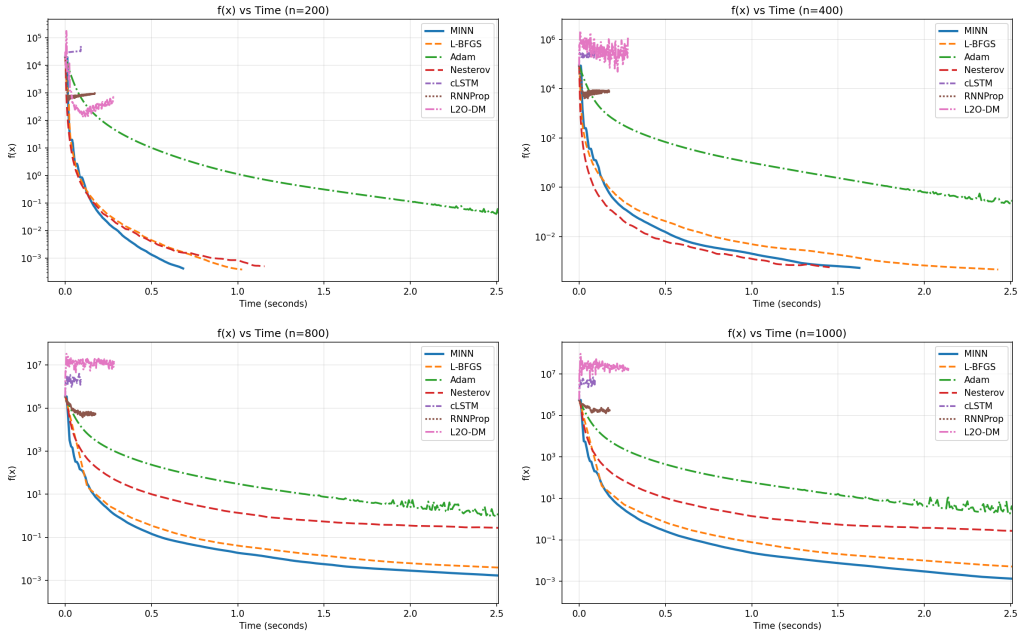

Figure 3: Random least-squares (OOD): convergence of $f(\boldsymbol{x}_t)$ vs. wall-clock time.

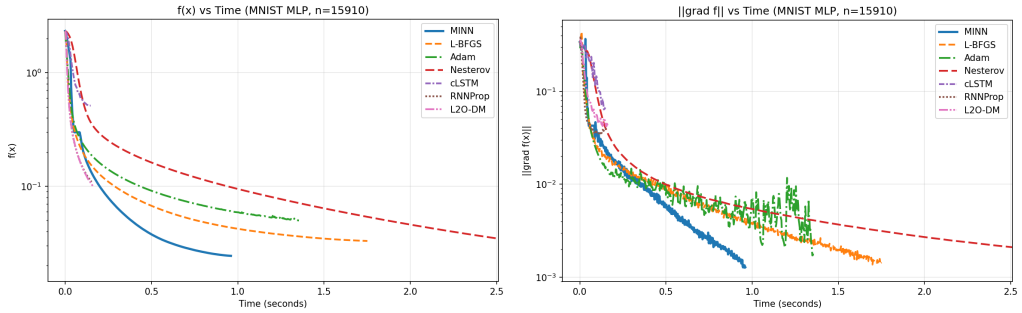

Figure 4: MNIST MLP training (in-distribution): training loss $f(\boldsymbol{x}_t)$ (left) and $\|\nabla f(\boldsymbol{x}_t)\|$ (right).

curvature adaptation, and the classical safeguards—half-space projection, Armijo backtracking, and the endpoint acceptance test—supply numerical stabilization.

Out-of-distribution. We next shift the test distribution to $A_{ij} \sim \mathcal{N}(0.15, 1)$ while keeping the training distribution fixed at $\mathcal{N}(0, 1)$. This evaluates robustness to distributional mismatch in addition to scale generalization. Figure 3 reports the corresponding convergence curves.

The distribution shift amplifies the separation between learned and hybrid learned–classical optimization. cLSTM, RNNProp, and L2O-DM lose stability almost immediately, indicating that their learned update rules do not provide reliable OOD descent. Classical methods remain robust, but their efficiency varies: first-order methods can be slow to reach high accuracy, and L-BFGS can require substantially more wall-clock time at larger dimensions. MINN remains stable under both scale and distribution shift, generalizing from $n_{\text{train}} = 200$ to $n_{\text{test}} \in \{400, 800, 1000\}$ under the shifted distribution. Its curves retain the early-time acceleration of learned optimizers while avoiding their OOD failures, yielding low objective values faster than the classical baselines except at n=400.

5.2 Experiment 2: Neural Network Training (MNIST MLP)

In-distribution. On this high-dimensional (n=15,910) task, we run each optimizer from 50 held-out initializations $\boldsymbol{x}_0$. Figure 4 shows convergence trajectories.

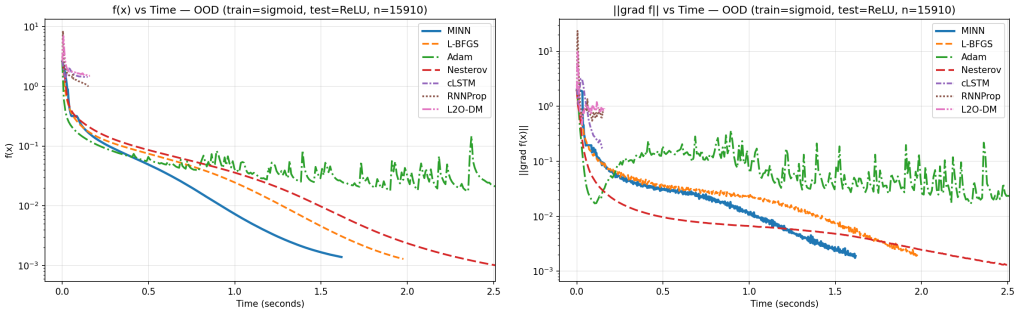

Figure 5: As Figure 4, but OOD: train on sigmoid, test on ReLU.

Out-of-distribution: sigmoid → ReLU transfer. To test OOD generalization, all optimizers trained on the sigmoid MLP are evaluated on the same architecture with ReLU activation—a significant change in loss landscape geometry. The ReLU objective is nonsmooth, so this transfer experiment is outside the scope of Propositions 1 and 2. At nondifferentiable points, the reported gradient norm is the norm of the vector returned by automatic differentiation; it is a diagnostic, not a nonsmooth stationarity certificate. Figure 5 shows convergence trajectories.

The MNIST MLP results reveal the same qualitative behavior as in the synthetic experiments. In the matched sigmoid setting, cLSTM, RNNProp, and L2O-DM reduce the loss rapidly at the beginning, confirming the benefit of amortized optimization. However, their curves quickly plateau at substantially worse loss and gradient norm than the classical baselines. Classical methods are slower initially but continue improving, with Nesterov eventually reaching the lowest loss (Table 10 in the appendix).

MINN combines these two regimes. It inherits the early-time acceleration of the learned optimizers, but its L-BFGS correction prevents the plateau behavior and allows continued progress toward a high-quality stationary point. Under the sigmoid-to-ReLU transfer test, the contrast becomes sharper: purely learned optimizers fail to transfer and remain at high loss, whereas classical methods and MINN remain stable. MINN achieves this robustness while retaining faster convergence than the classical baselines. These results indicate that MINN's learned component provides useful task-level acceleration, while the classical component supplies the numerical structure needed for OOD robustness and final accuracy.

6 Conclusion

We presented **MINN**, a hybrid optimizer that combines the adaptivity of learned optimization with the stability of classical safeguards. By separating neural *proposal* from classical *certification*, MINN achieves fast amortized progress while preserving monotone decrease and first-order convergence under the standard assumptions. Across least-squares, neural network training, and Rastrigin benchmarks, MINN improves the speed-accuracy trade-off and avoids the stalling and out-of-distribution failures of purely learned optimizers. These results suggest that embedding learning inside mathematically controlled optimization algorithms is a promising direction for robust learned optimization.

Reproducibility Statement

The benchmark formulations, implementation details, and additional aggregate results are documented in Appendices A–K. The accompanying anonymous supplementary package is intended to include the exact data split, trained checkpoints, evaluator, and scripts needed to reproduce every reported table and figure. Code will be made available upon acceptance of the paper.

AI Use Statement

Generative AI tools were used to assist with manuscript organization and drafting, literature search, figure and table formatting, feedback on experimental presentation, and inspection of code and experimental artifacts during preparation of this work. All AI-assisted material was reviewed by the authors. The authors verified the equations, claims, citations, tables, figures, and reported experimental results and take responsibility for the final content of the submission.

References

Brandon Amos. Tutorial on amortized optimization, 2025. URL https://arxiv.org/abs/2202.00665.

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3988–3996, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Sebastian Banert, Axel Ringh, Jonas Adler, Johan Karlsson, and Ozan Öktem. Data-driven nonsmooth optimization. *SIAM Journal on Optimization*, 30(1):102–131, 2020. doi: 10.1137/18M1207685.

Sebastian Banert, Jevgenija Rudzusika, Ozan Öktem, and Jonas Adler. Accelerated forward–backward optimization using deep learning. *SIAM Journal on Optimization*, 34(2):1236–1263, 2024. doi: 10.1137/22M1532548. URL https://arxiv.org/abs/2105.05210.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. Symbolic discovery of optimization algorithms. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=ne6zeqLFCZ.

Christian Daniel, Jonathan Taylor, Sebastian Nowozin, and Drew Bagnell. Learning step size controllers for robust neural network training. In *AAAI Conference on Artificial Intelligence*, 2016. URL https://arxiv.org/abs/1601.05077.

Patrick Fahy, Mohammad Golbabaee, and Matthias J. Ehrhardt. Greedy learning to optimize with convergence guarantees. arXiv preprint arXiv:2406.00260, 2024. URL https://arxiv.org/abs/2406.00260.

Xi Gao, Jinxin Xiong, Akang Wang, Jiang Xue, Qingjiang Shi, et al. IPM-LSTM: A learning-based interior point method for solving nonlinear programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 122891–122916, 2024.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, 2010.

Howard Heaton, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Safeguarded learned convex optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):7848–7855, 2023. doi: 10.1609/aaai.v37i6.25950. URL https://arxiv.org/abs/2003.01880.

Paul Häusner, Ozan Öktem, and Jens Sjölund. Neural incomplete factorization: learning preconditioners for the conjugate gradient method. arXiv preprint arXiv:2305.16368, 2023. URL https://arxiv.org/abs/2305.16368.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. URL https://arxiv.org/abs/1412.6980.

Isaac Liao, Rumen R. Dangovski, Jakob N. Foerster, and Marin Soljačić. Learning to optimize quasi-Newton methods. arXiv preprint arXiv:2210.06171, 2023. URL https://arxiv.org/abs/2210.06171.

Jialin Liu, Xiaohan Chen, Zhangyang Wang, Wotao Yin, and Hanqin Cai. Towards constituting mathematical structures for learning to optimize. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 21426–21449. PMLR, 2023. URL https://proceedings.mlr.press/v202/liu23e.html.

Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning (ICML)*, 2017. URL https://arxiv.org/abs/1703.03633.

Tim Meinhardt, Michael Möller, Caner Hazirbas, and Daniel Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *International Conference on Computer Vision (ICCV)*, 2017. URL https://arxiv.org/abs/1707.06474.

Michael Moeller, Thomas Möllenhoff, and Daniel Cremers. Controlling neural networks via energy dissipation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3255–3264, 2019. doi: 10.1109/ICCV.2019.00335. URL https://arxiv.org/abs/1904.03081.

Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021. doi: 10.1109/MSP.2020.3016905.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

Hoang T. Nguyen and Priya L. Donti. FSNet: Feasibility-seeking neural network for constrained optimization with guarantees, 2025. URL https://arxiv.org/abs/2506.00362.

J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN 9780387400655. URL https://books.google.com/books?id=VbHYoSyelFcC.

Isabeau Prémont-Schwarz, Jaroslav Vítků, and Jan Feyereisl. A simple guard for learned optimizers. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17910–17925. PMLR, 2022. URL https://proceedings.mlr.press/v162/premont-schwarz22a.html.

Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Narain Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. URL https://arxiv.org/abs/1703.04813.

Junzi Zhang, Brendan O'Donoghue, and Stephen Boyd. Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020. doi: 10.1137/18M1232772.

A NOTATION

Table 1 collects the symbols of Section 3. Within the method, the algorithms, and the statements of the main text, each symbol keeps the meaning given here. A few letters are reused locally, with their scope stated where they appear: the LSTM gates, weights, and states $\boldsymbol{f}, \boldsymbol{i}, \boldsymbol{o}, \boldsymbol{c}, W_\cdot, U_\cdot, \boldsymbol{b}_\cdot$ and the input $\boldsymbol{u}$ in Appendix C; the two-loop auxiliaries $\boldsymbol{q}, \boldsymbol{r}, a_i, b_i$ in Algorithm 2; generic test vectors $\boldsymbol{u}$ in Appendix D.1; the parameters of the benchmark problems; training bookkeeping variables in Algorithm 3; and auxiliary scalars inside the proofs.

Decorations. A tilde marks a value before transformation, such as $\tilde{\boldsymbol{d}}_t$ before projection or $\tilde{\mu}_k$ before $\mathrm{softplus}$. The hat in $\hat{\boldsymbol{x}}_t$ marks the Stage-1 reference; in $\widehat{\boldsymbol{x}}_\vartheta$ and $\widehat{\boldsymbol{g}}_\Theta$ it marks a prediction or estimate. A bar denotes averaging, over coordinates in $\bar{\boldsymbol{h}}_t$ and over training instances in $\overline{\mathcal{L}}$. The superscript n denotes rescaling to norm at most the current gradient norm. Superscripts (j) index coordinates and their states, whereas (i) in $\boldsymbol{W}^{(i)}$ counts curvature updates.

B ROLE OF EACH COMPONENT

Table 2 summarizes the components. The neural block proposes updates informed by training, while projection and Armijo backtracking produce a point with sufficient decrease. The curvature-based Stage-2 block then proposes further progress; its endpoint filter preserves the decrease already obtained by Stage 1.

C DETAILS OF COORDINATE-WISE LSTM BLOCK $\mathcal{F}_\theta$

We adopt the coordinate-wise LSTM architecture of Andrychowicz et al. (2016) with a *single layer*. With H (trial steps) and D (hidden-state dimension) as in Section 3.1, a single shared LSTM cell is unrolled for the H trial steps. Starting from $\boldsymbol{x}_t$, this trial run is used to produce the estimated direction $\boldsymbol{d}_t^{\mathrm{lstm}}$. Let $\boldsymbol{x}_{t,0}^{\mathrm{lstm}} = \boldsymbol{x}_t$. At outer iteration t, the LSTM processes each coordinate $j = 1, \dots, n$ with *shared* weights θ across coordinates and trial steps, but separate hidden and cell states. At trial step $h = 0, \dots, H-1$, the cell takes the two-dimensional input $\big[x_{t,h}^{\mathrm{lstm},(j)},\ g_{t,h}^{\mathrm{lstm},(j)}\big]$ and produces $\big[x_{t,h+1}^{\mathrm{lstm},(j)},\ d_{t,h}^{\mathrm{lstm},(j)}\big]$.

Input features at trial step h. For coordinate j we construct

$$\boldsymbol{u}_{t,h}^{(j)} = \big[x_{t,h}^{\mathrm{lstm},(j)},\ g_{t,h}^{\mathrm{lstm},(j)}\big]^\top \in \mathbb{R}^2, \tag{19}$$

where $\boldsymbol{g}_{t,h}^{\mathrm{lstm}} = \nabla f\big(\boldsymbol{x}_{t,h}^{\mathrm{lstm}}\big)$.

LSTM update at trial step h. The recurrent steps are chained within outer iteration t: the hidden state $\boldsymbol{h}_{t,h}^{(j)} \in \mathbb{R}^D$ and cell state $\boldsymbol{c}_{t,h}^{(j)} \in \mathbb{R}^D$ at step h are computed from the states at step $h-1$ and the current input. For the boundary condition at $h = 0$ we carry over the states from the last trial step of the previous outer iteration: $\boldsymbol{h}_{t,-1}^{(j)} \coloneqq \boldsymbol{h}_{t-1,H-1}^{(j)}$ and $\boldsymbol{c}_{t,-1}^{(j)} \coloneqq \boldsymbol{c}_{t-1,H-1}^{(j)}$ (initialized to $\mathbf{0}$ at $t = 0$). The standard LSTM equations at step h ($h = 0, \dots, H-1$) are:

$$\boldsymbol{f}_{t,h}^{(j)} = \sigma\big(W_f\, \boldsymbol{u}_{t,h}^{(j)} + U_f\, \boldsymbol{h}_{t,h-1}^{(j)} + \boldsymbol{b}_f\big), \tag{20}$$

$$\boldsymbol{i}_{t,h}^{(j)} = \sigma\big(W_i\, \boldsymbol{u}_{t,h}^{(j)} + U_i\, \boldsymbol{h}_{t,h-1}^{(j)} + \boldsymbol{b}_i\big), \tag{21}$$

$$\tilde{\boldsymbol{c}}_{t,h}^{(j)} = \tanh\big(W_c\, \boldsymbol{u}_{t,h}^{(j)} + U_c\, \boldsymbol{h}_{t,h-1}^{(j)} + \boldsymbol{b}_c\big), \tag{22}$$

$$\boldsymbol{c}_{t,h}^{(j)} = \boldsymbol{f}_{t,h}^{(j)} \odot \boldsymbol{c}_{t,h-1}^{(j)} + \boldsymbol{i}_{t,h}^{(j)} \odot \tilde{\boldsymbol{c}}_{t,h}^{(j)}, \tag{23}$$

$$\boldsymbol{o}_{t,h}^{(j)} = \sigma\big(W_o\, \boldsymbol{u}_{t,h}^{(j)} + U_o\, \boldsymbol{h}_{t,h-1}^{(j)} + \boldsymbol{b}_o\big), \tag{24}$$

$$\boldsymbol{h}_{t,h}^{(j)} = \boldsymbol{o}_{t,h}^{(j)} \odot \tanh\big(\boldsymbol{c}_{t,h}^{(j)}\big). \tag{25}$$

Table 1: Notation. Equation numbers point to the definitions.

Symbol	Meaning
Indices	
t	outer iteration
$h = 0, \dots, H-1$	LSTM trial step within outer iteration t
$k = 0, \dots, K-1$	L-BFGS step within outer iteration t
$i = 1, \dots, m_k$	position in the Stage-2 memory, $i = 1$ oldest
$j = 1, \dots, n$	coordinate
Points	
$\boldsymbol{x}_t$, $\boldsymbol{g}_t$	iterate and its gradient; the only points MINN returns
$\boldsymbol{x}_{t,h}^{\mathrm{lstm}}$, $\boldsymbol{g}_{t,h}^{\mathrm{lstm}}$	LSTM trial point, $h = 0, \dots, H$, with $\boldsymbol{x}_{t,0}^{\mathrm{lstm}} = \boldsymbol{x}_t$; gradient for $h \le H-1$
$\boldsymbol{h}_{t,h}^{(j)}$	LSTM hidden state of coordinate j after trial step h; the states at $h = H-1$ are carried to iteration $t+1$
$\hat{\boldsymbol{x}}_t$	Stage-1 point (10); fallback for $\boldsymbol{x}_{t+1}$ and reference point for Stage 2
$\boldsymbol{z}_{t,k}$, $\boldsymbol{g}_{t,k}$	refinement point, $\boldsymbol{z}_{t,0} = \hat{\boldsymbol{x}}_t$, and its gradient
$\boldsymbol{z}_t$	Stage-2 endpoint (last refinement point), candidate for $\boldsymbol{x}_{t+1}$ (13)
Stage 1	
$\boldsymbol{d}_t^{\mathrm{lstm}}$	LSTM direction: last trial increment (2)
$\boldsymbol{s}_h^{\mathrm{lstm}}$, $\boldsymbol{y}_h^{\mathrm{lstm}}$	trial-run displacement and gradient change, $h = H-m, \dots, H-1$ (3)
$\mathcal{F}_\theta$	trial-run map from $\boldsymbol{x}_t$ and the carried states to $\boldsymbol{d}_t^{\mathrm{lstm}}$, $\bar{\boldsymbol{h}}_t$, and the trial-run pairs
$\boldsymbol{d}_t^{\mathrm{lstm,n}}$	$\boldsymbol{d}_t^{\mathrm{lstm}}$ rescaled to norm at most $\|\boldsymbol{g}_t\|$ (6)
$\bar{\boldsymbol{h}}_t$	final LSTM hidden states averaged over coordinates (4)
τ_t, ν_t	trust-head weights on $-\boldsymbol{g}_t$ and on $\boldsymbol{d}_t^{\mathrm{lstm,n}}$ (5)
$\tau_{\min}, \tau_{\max}$; $\nu_{\max}$; σ	bounds on τ_t; bound on ν_t; logistic sigmoid (5)
$\tilde{\boldsymbol{d}}_t$	fused proposal, before projection (6)
$\mathcal{H}_t$, κ	sufficient-descent half-space and its constant (7)
$\boldsymbol{d}_t$	certified direction: projection of $\tilde{\boldsymbol{d}}_t$ onto $\mathcal{H}_t$ (8)
η_t; c_1, β	Armijo step; sufficient-decrease constant and backtracking factor (9)
Stage 2	
$\mathcal{S}_\psi$	Stage-2 map $(\hat{\boldsymbol{x}}_t, \mathcal{M}_0) \mapsto \boldsymbol{z}_t$ (Algorithm 2)
$\mathcal{M}_0$, m_0	seed memory: the trial-run pairs (3) with $\boldsymbol{y}^\top \boldsymbol{s} > \varepsilon_{\mathrm{curv}}$, and its size $m_0 \le m$
$\mathcal{M}_k$, m_k	memory before inner step k and its size, $m_k \le m$
$(\boldsymbol{s}_i, \boldsymbol{y}_i)$	displacement and gradient difference stored at position i
$\boldsymbol{v}$	learned logits of the memory slots (11)
π_i, ρ_i	slot weight and update coefficient (11)
γ_k	initial inverse-Hessian scale from the newest pair, when $m_k > 0$ (Appendix D.1)
$\boldsymbol{W}^{(i)}$	approximation after the update with pair i, $\boldsymbol{W}^{(0)} = \gamma_k \boldsymbol{I}$ for $m_k > 0$ (26)
$\boldsymbol{W}_k$, $\boldsymbol{p}_k$	inverse-Hessian approximation $\boldsymbol{W}_k = \boldsymbol{W}^{(m_k)}$ for $m_k > 0$, otherwise $\boldsymbol{I}$; $\boldsymbol{p}_k = -\boldsymbol{W}_k \boldsymbol{g}_{t,k}$
μ_k	learned step length of inner step k (12)
Parameters and constants	
θ, ϕ, ψ; Θ	LSTM, trust head, Stage-2 scalars $\psi = \{\boldsymbol{v}, \tilde{\boldsymbol{\mu}}\}$; $\Theta = (\theta, \phi, \psi)$
$\Theta^\star$	trained parameters, fixed at test time (Algorithm 4)
H, D, m, K	trial steps, hidden width, memory capacity, maximum inner steps
$\varepsilon_{\mathrm{dir}}, \varepsilon_{\mathrm{den}}, \varepsilon_{\mathrm{curv}}, \varepsilon_{\mathrm{tol}}$	normalization guard, division guard, curvature threshold, gradient tolerance
T	number of outer iterations unrolled in training (15)
$\xi \sim \mathcal{D}$; $\mathcal{C}$, d_ξ	instance context and its distribution; context domain and dimension (1)
ϑ	parameters of the direct amortized model in the overview
$\mathcal{P}_0(\cdot \mid \xi)$	task-specific distribution of the initial point $\boldsymbol{x}_0$ (16)
$\widehat{\boldsymbol{g}}_\Theta$	TBPTT estimate of the batch meta-gradient (Algorithm 3)
ω_t, λ	iteration weights and stationarity weight of the loss (15)
T_{bp}	truncation window of TBPTT (Appendix E)
L	Lipschitz constant of ∇f (Section 4)
$\mathcal{X}_0$	level set $\{\boldsymbol{x} : f(\boldsymbol{x}) \le f(\boldsymbol{x}_0)\}$ of (A1)
C_d, $\eta_{\min}$	$\|\boldsymbol{d}_t\| \le C_d \|\boldsymbol{g}_t\|$ and lower bound on η_t (Appendix G.2)

Table 2: Components of MINN and their roles.

Component	Type	Role
Coordinate-wise LSTM $\mathcal{F}_\theta$	Neural	Propose adaptive, history-aware direction
Steepest Descent $-\boldsymbol{g}_t$	Classical	Provide reliable baseline direction
Bounded Scalar Trust Head (τ_t, ν_t)	Neural	Balance gradient backbone and learned correction
Projection Safeguard (8)	Classical safeguard	Enforce sufficient descent with minimal correction
Armijo Line Search (9)	Classical safeguard	Ensure sufficient decrease of the Stage-1 step
LSTM Curvature Pairs (3)	Neural→Classical	Bootstrap L-BFGS memory from LSTM trajectory
Learnable L-BFGS $\mathcal{S}_\psi$	Hybrid	Exploit curvature; propose Stage-2 refinement
Accept/Reject Filter (13)	Classical safeguard	Preserve monotonicity of the full two-stage method

Outputs. The hidden state (25) is read out as $d_{t,h}^{\mathrm{lstm},(j)} = \boldsymbol{w}_d^\top \boldsymbol{h}_{t,h}^{(j)} + b_d$, which advances the trial point by $x_{t,h+1}^{\mathrm{lstm},(j)} = x_{t,h}^{\mathrm{lstm},(j)} + d_{t,h}^{\mathrm{lstm},(j)}$, as in Section 3.1. The trial run then yields the direction (2), the last m curvature pairs (3) (assuming $1 \le m \le H-1$), and the pooled state (4); these are defined in Section 3.1 and not restated here. The pairs that pass $\boldsymbol{y}^\top \boldsymbol{s} > \varepsilon_{\mathrm{curv}}$, relabelled so that $i = 1$ is the oldest, form the seed memory $\mathcal{M}_0$ of Stage 2 ($m_0 \le m$).

Finally, the output of the coordinate-wise LSTM block is

$$\big(\boldsymbol{d}_t^{\mathrm{lstm}},\ \bar{\boldsymbol{h}}_t,\ \{\boldsymbol{s}_h^{\mathrm{lstm}}\}_{h=H-m}^{H-1},\ \{\boldsymbol{y}_h^{\mathrm{lstm}}\}_{h=H-m}^{H-1}\big) = \mathcal{F}_\theta(\boldsymbol{x}_t, H),$$

where $\mathcal{F}_\theta$ implicitly depends on the LSTM hidden and cell states carried over from the previous outer iteration ($\boldsymbol{h}_{t-1,H-1}^{(j)}, \boldsymbol{c}_{t-1,H-1}^{(j)}$ for all j). The full set of LSTM parameters is

$$\theta = \{W_f, U_f, \boldsymbol{b}_f,\ W_i, U_i, \boldsymbol{b}_i,\ W_c, U_c, \boldsymbol{b}_c,\ W_o, U_o, \boldsymbol{b}_o,\ \boldsymbol{w}_d, b_d\}.$$

The parameters of the scalar trust head are introduced in Section 3.2.

Parameter sharing. Because the weights are shared across all n coordinates, the number of learnable parameters is $O(D^2)$ and is *independent of the problem dimension* n.

D Learnable L-BFGS Pseudocode

Algorithm 2 performs at most K inner steps, with an early return when the gradient tolerance is reached. The outer index t is fixed during the call: $\boldsymbol{z}_{t,k}$ is its kth refinement point and $\boldsymbol{g}_{t,k} = \nabla f(\boldsymbol{z}_{t,k})$. The starting point is $\hat{\boldsymbol{x}}_t$; the output $\boldsymbol{z}_t$ is the last refinement point reached, before the outer acceptance test. The index t is suppressed only on the memory, its coefficients, and the search directions. The objective is fixed during the call; in training it is $f(\cdot; \xi_b)$. Memory slots are ordered from oldest ($i = 1$) to newest ($i = m_k$). The trained logits $\boldsymbol{v}$ weight these slots, and the trained $\tilde{\mu}_k$ controls the step at inner index k; both are reused across calls. The seed memory contains only the trial-run pairs that pass the curvature test, and newly generated inner-step pairs must pass the same test.

D.1 Weighted BFGS Operator and Descent Directions

The two-loop recursion in Algorithm 2 applies a linear operator $\boldsymbol{W}_k$ to the current gradient $\boldsymbol{g}_{t,k}$ and returns the direction $\boldsymbol{p}_k = -\boldsymbol{W}_k \boldsymbol{g}_{t,k}$. The matrices below describe this operator; they are not formed or stored by the algorithm. Fix an inner step k with nonempty memory, ordered from oldest to newest, and start from $\boldsymbol{W}^{(0)} = \gamma_k \boldsymbol{I}$, where

$$\gamma_k = \frac{\boldsymbol{s}_{m_k}^\top \boldsymbol{y}_{m_k}}{\boldsymbol{y}_{m_k}^\top \boldsymbol{y}_{m_k} + \varepsilon_{\mathrm{den}}}.$$

For the coefficients ρ_i in (11), the recursion represents the successive inverse-BFGS-form updates

$$\boldsymbol{W}^{(i)} = (\boldsymbol{I} - \rho_i \boldsymbol{s}_i \boldsymbol{y}_i^\top)\, \boldsymbol{W}^{(i-1)} (\boldsymbol{I} - \rho_i \boldsymbol{y}_i \boldsymbol{s}_i^\top) + \rho_i \boldsymbol{s}_i \boldsymbol{s}_i^\top, \qquad i = 1, \dots, m_k, \tag{26}$$

with $\boldsymbol{W}_k = \boldsymbol{W}^{(m_k)}$. The standard two-loop recursion evaluates $\boldsymbol{W}_k \boldsymbol{g}_{t,k}$ in $O(m_k n)$ operations for any values of the scalars ρ_i. With empty memory, the algorithm instead uses $\boldsymbol{W}_k = \boldsymbol{I}$ and $\boldsymbol{p}_k = -\boldsymbol{g}_{t,k}$.

Algorithm 2 Stage-2 refinement $\mathcal{S}_\psi$: LearnableLBFGS($\hat{\boldsymbol{x}}_t, K, \mathcal{M}_0, m, \varepsilon_{\text{tol}}, \boldsymbol{v}, \tilde{\boldsymbol{\mu}}$)

Require: Certified starting point $\hat{\boldsymbol{x}}_t$, maximum inner steps K, initial memory $\mathcal{M}_0 = \{(\boldsymbol{s}_i, \boldsymbol{y}_i)\}_{i=1}^{m_0}$, $m_0 \le m$: the seed pairs of (3) with $\boldsymbol{y}^\top \boldsymbol{s} > \varepsilon_{\text{curv}}$, relabelled in order so that $i = 1$ is the oldest, learnable logits $\boldsymbol{v} \in \mathbb{R}^m$, learnable step-size parameters $\tilde{\boldsymbol{\mu}} = (\tilde{\mu}_0, \dots, \tilde{\mu}_{K-1})$, memory capacity m, gradient tolerance ε_{tol}. Fixed constants: division guard $\varepsilon_{\text{den}} = 10^{-10}$ and curvature threshold $\varepsilon_{\text{curv}} = 10^{-10}$ (Table 3)

Ensure: Refinement endpoint $\boldsymbol{z}_t$ for the outer acceptance test; the final memory is discarded

$\boldsymbol{z}_{t,0} \leftarrow \hat{\boldsymbol{x}}_t$; $\boldsymbol{g}_{t,0} \leftarrow \nabla f(\boldsymbol{z}_{t,0})$; $\mathcal{M} \leftarrow \mathcal{M}_0$
for $k = 0, 1, \dots, K-1$ **do**
  $\mu_k \leftarrow \text{softplus}(\tilde{\mu}_k)$
  $m_k \leftarrow |\mathcal{M}|$
  if $m_k = 0$ **then**
    $\boldsymbol{p}_k \leftarrow -\boldsymbol{g}_{t,k}$ *// Fallback to steepest descent*
  else
    $\boldsymbol{\pi} \leftarrow \text{softmax}(\boldsymbol{v}_{1:m_k})$ *// Slot weights, computed from the learned logits $\boldsymbol{v}$*
    for $i = 1, \dots, m_k$ **do**
      $\rho_i \leftarrow \pi_i / (\boldsymbol{y}_i^\top \boldsymbol{s}_i + \varepsilon_{\text{den}})$
    end for
    // Two-loop recursion
    $\boldsymbol{q} \leftarrow \boldsymbol{g}_{t,k}$
    for $i = m_k, m_k-1, \dots, 1$ **do**
      $a_i \leftarrow \rho_i \boldsymbol{s}_i^\top \boldsymbol{q}$; $\boldsymbol{q} \leftarrow \boldsymbol{q} - a_i \boldsymbol{y}_i$
    end for
    $\gamma_k \leftarrow \boldsymbol{s}_{m_k}^\top \boldsymbol{y}_{m_k} / (\boldsymbol{y}_{m_k}^\top \boldsymbol{y}_{m_k} + \varepsilon_{\text{den}})$; $\boldsymbol{r} \leftarrow \gamma_k \boldsymbol{q}$
    for $i = 1, 2, \dots, m_k$ **do**
      $b_i \leftarrow \rho_i \boldsymbol{y}_i^\top \boldsymbol{r}$; $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{s}_i (a_i - b_i)$
    end for
    $\boldsymbol{p}_k \leftarrow -\boldsymbol{r}$
  end if
  $\boldsymbol{z}_{t,k+1} \leftarrow \boldsymbol{z}_{t,k} + \mu_k \boldsymbol{p}_k$ *// Eq. (12)*
  $\boldsymbol{g}_{t,k+1} \leftarrow \nabla f(\boldsymbol{z}_{t,k+1})$
  $\boldsymbol{s}_{\text{new}} \leftarrow \boldsymbol{z}_{t,k+1} - \boldsymbol{z}_{t,k}$; $\boldsymbol{y}_{\text{new}} \leftarrow \boldsymbol{g}_{t,k+1} - \boldsymbol{g}_{t,k}$
  if $\boldsymbol{y}_{\text{new}}^\top \boldsymbol{s}_{\text{new}} > \varepsilon_{\text{curv}}$ **then**
    Append $(\boldsymbol{s}_{\text{new}}, \boldsymbol{y}_{\text{new}})$ to $\mathcal{M}$
    if $|\mathcal{M}| > m$ **then**
      Remove oldest pair from $\mathcal{M}$
    end if
  end if
  if $\|\boldsymbol{g}_{t,k+1}\| < \varepsilon_{\text{tol}}$ **then**
    $\boldsymbol{z}_t \leftarrow \boldsymbol{z}_{t,k+1}$; **return** $\boldsymbol{z}_t$
  end if
end for
$\boldsymbol{z}_t \leftarrow \boldsymbol{z}_{t,K}$; **return** $\boldsymbol{z}_t$

How the weights change a classical BFGS update. Classical L-BFGS takes $\rho_i = 1/(\boldsymbol{y}_i^\top \boldsymbol{s}_i)$, so that the update with pair i satisfies its secant equation, $\boldsymbol{W}^{(i)} \boldsymbol{y}_i = \boldsymbol{s}_i$. Later updates need not preserve this equation; in general, the final classical approximation satisfies it exactly only for the newest pair. MINN changes the update coefficients using the learned slot weights of (11). With $\varepsilon_{\text{den}} = 0$, we have $\rho_i \boldsymbol{s}_i^\top \boldsymbol{y}_i = \pi_i$, so $(\boldsymbol{I} - \rho_i \boldsymbol{y}_i \boldsymbol{s}_i^\top)\boldsymbol{y}_i = (1 - \pi_i)\boldsymbol{y}_i$ and $\rho_i \boldsymbol{s}_i \boldsymbol{s}_i^\top \boldsymbol{y}_i = \pi_i \boldsymbol{s}_i$. Applying (26) to $\boldsymbol{y}_i$ therefore gives

$$\boldsymbol{W}^{(i)} \boldsymbol{y}_i = \pi_i \boldsymbol{s}_i + (1 - \pi_i)(\boldsymbol{I} - \rho_i \boldsymbol{s}_i \boldsymbol{y}_i^\top) \boldsymbol{W}^{(i-1)} \boldsymbol{y}_i. \tag{27}$$

For $\varepsilon_{\text{den}} > 0$, the same identity holds with each explicit π_i in (27) replaced by $\pi_i \boldsymbol{y}_i^\top \boldsymbol{s}_i / (\boldsymbol{y}_i^\top \boldsymbol{s}_i + \varepsilon_{\text{den}})$. Thus the weight controls how strongly the pair changes the approximation: when the division guard is zero, $\pi_i = 1$ gives the classical BFGS update, whereas $\pi_i \to 0$ leaves $\boldsymbol{W}^{(i-1)}$ unchanged by that update. The final approximation generally need not satisfy even the newest secant equation exactly.

The logits are fixed after training and attached to positions in the current memory: the oldest pair receives v_1, the newest receives v_{m_k}, and $v_{m_k+1}, \ldots, v_m$ are unused when the memory is not full. They are not predicted from the numerical values of the pairs. At initialization ($\boldsymbol{v} = \mathbf{0}$), each slot has weight $1/m_k$. Because the softmax weights are positive and sum to one, each is strictly less than one when $m_k \geq 2$. Consequently, for a memory with multiple pairs, no update coefficient equals its classical value; for one pair, the coefficient coincides with the classical coefficient when $\varepsilon_{\text{den}} = 0$. This is a weighted variant of the L-BFGS recursion, rather than a parameterization containing classical L-BFGS for every memory size.

Why the direction is a descent direction. Every stored pair has $\boldsymbol{y}_i^\top \boldsymbol{s}_i > \varepsilon_{\text{curv}} > 0$, whether it came from the LSTM trial run or from refinement. Therefore $\gamma_k > 0$ and $\rho_i > 0$. Each update (26) preserves symmetric positive definiteness: for a test vector $\boldsymbol{u}$, let $\boldsymbol{u}_i = (\boldsymbol{I} - \rho_i \boldsymbol{y}_i \boldsymbol{s}_i^\top)\boldsymbol{u}$. Then

$$\boldsymbol{u}^\top \boldsymbol{W}^{(i)} \boldsymbol{u} = \boldsymbol{u}_i^\top \boldsymbol{W}^{(i-1)} \boldsymbol{u}_i + \rho_i (\boldsymbol{s}_i^\top \boldsymbol{u})^2.$$

If $\boldsymbol{W}^{(i-1)} \succ 0$, both terms are nonnegative. If both vanish, then $\boldsymbol{s}_i^\top \boldsymbol{u} = 0$, hence $\boldsymbol{u}_i = \boldsymbol{u}$, and the first term implies $\boldsymbol{u} = \mathbf{0}$. Starting from $\boldsymbol{W}^{(0)} = \gamma_k \boldsymbol{I} \succ 0$ therefore gives $\boldsymbol{W}_k \succ 0$. The same property holds with empty memory, when $\boldsymbol{W}_k = \boldsymbol{I}$. Consequently,

$$\boldsymbol{g}_{t,k}^\top \boldsymbol{p}_k = -\boldsymbol{g}_{t,k}^\top \boldsymbol{W}_k \boldsymbol{g}_{t,k} < 0 \qquad \text{when } \boldsymbol{g}_{t,k} \neq \mathbf{0}.$$

This is a statement about the direction at $\boldsymbol{z}_{t,k}$. A positive learned step length need not decrease f, so it does not replace the endpoint acceptance test (13). The convergence results for the outer iterates use that test, not a convergence guarantee for the inner weighted L-BFGS run.

E Increasing-Horizon Curriculum Training

Full unrolling for large T is memory-intensive and suffers from exploding or vanishing meta-gradients. We adopt an *increasing-horizon curriculum*:

1. Start with a short horizon $T_{\min}$ (Table 3 lists the values used).
2. Train until the meta-loss plateaus.
3. Increase: $T \leftarrow \min(T + \Delta T,\ T_{\max}^{\text{train}})$.
4. Repeat steps 2–3 until $T = T_{\max}^{\text{train}}$, then train at $T_{\max}^{\text{train}}$ until the meta-loss plateaus.

Additionally, we use *truncated backpropagation through time* (TBPTT) with window size T_{bp}: meta-gradients are approximated by truncating dependencies across windows of outer iterations. Exact meta-gradients would retain dependencies through both accepted iterates and recurrent states. The truncated estimate need not equal the gradient of the full unroll.

Training details.

- Meta-optimizer: Adam with learning rate 10^{-3} and cosine annealing.
- Gradient clipping: global norm clipping at 1.0.
- Initialization: LSTM weights from Xavier; L-BFGS logits $\boldsymbol{v}$ at $\mathbf{0}$ (uniform weighting); $\tilde{\mu}_k$ at $\text{softplus}^{-1}(1)$ (unit step size); trust-head weights and biases near zero so that $\tau_t \approx (\tau_{\min} + \tau_{\max})/2$ and $\nu_t \approx \nu_{\max}/2$ initially.

Table 3 lists the hyperparameters used in all experiments.

Evaluation Environment and Metrics. All experiments are conducted on a system with an NVIDIA H100 GPU and an AMD EPYC 9634 3.7 GHz CPU, using Python 3.10 and PyTorch 1.13.1. Baselines are used with default settings. We report mean best objective value $f(\boldsymbol{x})$, mean best gradient norm $\|\nabla f(\boldsymbol{x})\|$, and mean wall-clock time (seconds) at termination. Convergence trajectories plot $f(\boldsymbol{x}_t)$ and $\|\nabla f(\boldsymbol{x}_t)\|_2$ versus wall-clock time, averaged over test instances.

Table 3: MINN hyperparameters.

Parameter	**Value**
LSTM trial steps H	30
LSTM hidden dimension D	10
L-BFGS memory m	5
L-BFGS inner iterations K	20
Descent constant κ	10^{-3}
Armijo c_1	10^{-4}
Armijo backtrack β	0.5
Trust lower bound $\tau_{\min}$	10^{-2}
Trust upper bound $\tau_{\max}$	1.0
Fusion-weight bound $\nu_{\max}$	1.0
Normalization guard $\varepsilon_{\mathrm{dir}}$	10^{-8}
Division guard $\varepsilon_{\mathrm{den}}$	10^{-10}
Curvature threshold $\varepsilon_{\mathrm{curv}}$	10^{-10}
Meta-learning rate	10^{-3}
Batch size B	64
Curriculum $T_{\min}/\Delta T/T_{\max}^{\mathrm{train}}$	1 / 1 / 4
Stationarity weight λ	0.01
Test-time max iterations $T_{\max}^{\mathrm{test}}$	15000

E.1 Training Procedure Summary

Algorithm 3 uses the same proposal, fusion, refinement, and acceptance equations as inference and adjusts Θ using the trajectory loss; the inference procedure additionally tests the Stage-1 gradient norm before refinement. The initializer $\mathcal{P}_0(\cdot \mid \xi)$ is standard normal unless the task setup specifies otherwise. Checkpoints are compared only at a fixed horizon. After a horizon increase, at least one mini-batch is evaluated before any stopping decision; $\Theta^\star$ is the best observed checkpoint at the final trained horizon. Training stops after P meta-updates without improvement at $T = T_{\max}^{\mathrm{train}}$. $\mathrm{AdamStep}$ denotes the Adam update with global-norm clipping of $\widehat{\boldsymbol{g}}_\Theta$ at 1.0, base learning rate ℓ with cosine annealing, and moment estimates initialized to zero and retained across meta-updates. Within each instance, hidden and cell states carry over between outer iterations. Each Stage-2 call starts at $\hat{\boldsymbol{x}}_t$ with new LSTM-derived curvature memory; its final memory is discarded, and the next outer iteration's trial run starts at the accepted $\boldsymbol{x}_{t+1}$.

F Test-Time Optimization Procedure

At test time, Algorithm 4 applies MINN to a new instance with all trained parameters $\Theta^\star$ fixed. Hidden and cell states start at zero and carry information between outer iterations; Stage-2 curvature memory is reset on every call. The trust coefficients and curvature pairs are recomputed from the current trial run, while the learned memory logits and inner-step parameters remain fixed. The procedure is the outer iteration of Algorithm 1 with two stopping tests: it returns $\boldsymbol{x}_t$ once $\|\boldsymbol{g}_t\| < \varepsilon_{\mathrm{tol}}$, and it returns $\boldsymbol{x}_{t+1} = \hat{\boldsymbol{x}}_t$, without running Stage 2, once $\|\nabla f(\hat{\boldsymbol{x}}_t)\| < \varepsilon_{\mathrm{tol}}$. No meta-gradients are computed. Forward evaluations of the learned blocks, objective and gradient evaluations, Armijo backtracking, and the Stage-2 endpoint comparison are still required.

Computational cost per outer iteration. The counts below assume that $f(\boldsymbol{x}_t)$ and $\boldsymbol{g}_t$ are available and that repeated objective and gradient evaluations are cached.

- LSTM forward pass: $O(nHD^2)$ (linear in n).
- Pooled hidden-state summary: $O(nD)$.
- Scalar trust fusion, normalization, and projection: $O(n)$.
- Armijo line search: $O(n)$ per trial, with at most $1 + \lceil \log_{1/\beta}(1/\eta_{\min}) \rceil$ trials under the hypotheses of Proposition 1 (see Step 3 of its proof).

Algorithm 3 MINN: Offline Amortized Training with Stage-2 Accept/Reject Safeguard

Require: Problem distribution $\mathcal{D}$, initialization law $\mathcal{P}_0(\cdot \mid \xi)$, batch size B, TBPTT window T_{bp}, curriculum $(T_{\min}, \Delta T, T_{\max}^{\mathrm{train}})$, meta-learning rate ℓ, LSTM trial steps H, L-BFGS memory m, L-BFGS iterations K, patience P, tolerance $\varepsilon_{\mathrm{tol}}$, trust bounds $(\tau_{\min}, \tau_{\max}, \nu_{\max})$, descent constant κ, Armijo parameters c_1, β, loss weights ω_t and λ

Initialize $\Theta = (\theta, \phi, \psi)$ where $\phi = \{\boldsymbol{w}_\tau, b_\tau, \boldsymbol{w}_\nu, b_\nu\}$, $\psi = \{\boldsymbol{v}, \tilde{\boldsymbol{\mu}}\}$
$T \leftarrow T_{\min}$; $\Theta^\star \leftarrow \Theta$; $\mathcal{L}_{\mathrm{best}} \leftarrow +\infty$
while not converged **do**
  Sample mini-batch $\{\xi_b\}_{b=1}^{B} \sim \mathcal{D}$
  for each instance ξ_b **do**
    Initialize $\boldsymbol{x}_0 \sim \mathcal{P}_0(\cdot \mid \xi_b)$; set $\boldsymbol{h}_{0,-1}^{(j)} = \mathbf{0}$, $\boldsymbol{c}_{0,-1}^{(j)} = \mathbf{0}$ for $j = 1, \ldots, n$
    for $t = 0, \ldots, T-1$ **do**
      $\boldsymbol{g}_t \leftarrow \nabla f(\boldsymbol{x}_t; \xi_b)$
      if $\|\boldsymbol{g}_t\| < \varepsilon_{\mathrm{tol}}$ **then**
        // Tolerance reached: both stages skipped; corresponding loss terms use $\boldsymbol{x}_t$
        $\hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t$
        $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t$
        continue
      end if
      // Stage 1: Neural proposal, projection, and Armijo step
      $(\boldsymbol{d}_t^{\mathrm{lstm}}, \bar{\boldsymbol{h}}_t, \{\boldsymbol{s}_h^{\mathrm{lstm}}\}, \{\boldsymbol{y}_h^{\mathrm{lstm}}\}) \leftarrow \mathcal{F}_\theta(\boldsymbol{x}_t, H)$ *// Appendix C*
      $\tau_t \leftarrow \tau_{\min} + (\tau_{\max} - \tau_{\min})\sigma(\boldsymbol{w}_\tau^\top \bar{\boldsymbol{h}}_t + b_\tau)$
      $\nu_t \leftarrow \nu_{\max}\, \sigma(\boldsymbol{w}_\nu^\top \bar{\boldsymbol{h}}_t + b_\nu)$
      $\boldsymbol{d}_t^{\mathrm{lstm,n}} \leftarrow \frac{\|\boldsymbol{g}_t\|}{\|\boldsymbol{d}_t^{\mathrm{lstm}}\| + \varepsilon_{\mathrm{dir}}}\, \boldsymbol{d}_t^{\mathrm{lstm}}$
      $\tilde{\boldsymbol{d}}_t \leftarrow -\tau_t\, \boldsymbol{g}_t + \nu_t\, \boldsymbol{d}_t^{\mathrm{lstm,n}}$ *// Eq. (6)*
      $\boldsymbol{d}_t \leftarrow \tilde{\boldsymbol{d}}_t - \frac{\max\{0,\, \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa \|\boldsymbol{g}_t\|^2\}}{\|\boldsymbol{g}_t\|^2}\, \boldsymbol{g}_t$ *// Eq. (8)*
      $\eta_t \leftarrow \mathrm{ArmijoBacktrack}(f, \boldsymbol{x}_t, \boldsymbol{g}_t, \boldsymbol{d}_t, c_1, \beta)$ *// Eq. (9)*
      $\hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t + \eta_t\, \boldsymbol{d}_t$
      // Stage 2: Learnable L-BFGS refinement candidate
      $\mathcal{M}_0 \leftarrow \{(\boldsymbol{s}_h^{\mathrm{lstm}}, \boldsymbol{y}_h^{\mathrm{lstm}}) : H - m \le h \le H - 1,\ \boldsymbol{y}_h^{\mathrm{lstm}\top} \boldsymbol{s}_h^{\mathrm{lstm}} > \varepsilon_{\mathrm{curv}}\}$ *// Eq. (3)*
      $\boldsymbol{z}_t \leftarrow \mathrm{LearnableLBFGS}(\hat{\boldsymbol{x}}_t, K, \mathcal{M}_0, m, \varepsilon_{\mathrm{tol}}, \boldsymbol{v}, \tilde{\boldsymbol{\mu}})$ *// Algorithm 2*
      // Stage 2: Accept or reject the refinement endpoint
      if $f(\boldsymbol{z}_t; \xi_b) \le f(\hat{\boldsymbol{x}}_t; \xi_b)$ **then**
        $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{z}_t$
      else
        $\boldsymbol{x}_{t+1} \leftarrow \hat{\boldsymbol{x}}_t$
      end if
    end for
    $\mathcal{L}_b \leftarrow \sum_{t=1}^{T} \omega_t \big[f(\boldsymbol{x}_t; \xi_b) + f(\hat{\boldsymbol{x}}_{t-1}; \xi_b) + \lambda \big(\|\nabla f(\boldsymbol{x}_t; \xi_b)\|^2 + \|\nabla f(\hat{\boldsymbol{x}}_{t-1}; \xi_b)\|^2 \big) \big]$
  end for
  $\overline{\mathcal{L}} \leftarrow B^{-1} \sum_{b=1}^{B} \mathcal{L}_b$
  if $\overline{\mathcal{L}} < \mathcal{L}_{\mathrm{best}}$ **then**
    $\mathcal{L}_{\mathrm{best}} \leftarrow \overline{\mathcal{L}}$; $\Theta^\star \leftarrow \Theta$ *// Checkpoint the parameters just evaluated*
  end if
  Compute the TBPTT estimate $\widehat{\boldsymbol{g}}_\Theta$ of $\nabla_\Theta \overline{\mathcal{L}}$ using window T_{bp}
  $\Theta \leftarrow \mathrm{AdamStep}(\Theta, \widehat{\boldsymbol{g}}_\Theta; \ell)$
  If there is no improvement for P meta-updates and $T < T_{\max}^{\mathrm{train}}$, set $T \leftarrow \min(T + \Delta T, T_{\max}^{\mathrm{train}})$; reset $\mathcal{L}_{\mathrm{best}} \leftarrow +\infty$ and the no-improvement count; at $T = T_{\max}^{\mathrm{train}}$, P meta-updates without improvement end the loop (converged)
end while
return $\Theta^\star$

- L-BFGS two-loop recursion: $O(mn)$ per inner step $\times$ K iterations.
- Accept/reject monotonicity filter: one additional objective comparison after Stage 2.

Algorithm 4 MINN: Test-Time Optimization with Stage-2 Accept/Reject Safeguard

Require: Trained parameters $\Theta^\star = (\theta^\star, \phi^\star, \psi^\star)$, objective f, initial point $\boldsymbol{x}_0$, tolerance ε_{tol}, max iterations $T^{\text{test}}_{\max}$, LSTM trial steps H, L-BFGS memory m, L-BFGS iterations K, trust bounds $(\tau_{\min}, \tau_{\max}, \nu_{\max})$, descent constant κ, Armijo parameters c_1, β

Initialize $\boldsymbol{h}^{(j)}_{0,-1} = \mathbf{0}$, $\boldsymbol{c}^{(j)}_{0,-1} = \mathbf{0}$ for $j = 1, \dots, n$
for $t = 0, 1, \dots, T^{\text{test}}_{\max} - 1$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla f(\boldsymbol{x}_t)$
  if $\|\boldsymbol{g}_t\| < \varepsilon_{\text{tol}}$ **then**
    return $\boldsymbol{x}_t$ *// Converged*
  end if
  // — Stage 1: Neural proposal, projection, and Armijo step —
  $(\boldsymbol{d}^{\text{lstm}}_t, \bar{\boldsymbol{h}}_t, \{\boldsymbol{s}^{\text{lstm}}_h\}, \{\boldsymbol{y}^{\text{lstm}}_h\}) \leftarrow \mathcal{F}_{\theta^\star}(\boldsymbol{x}_t, H)$
  $\tau_t \leftarrow \tau_{\min} + (\tau_{\max} - \tau_{\min})\sigma(\boldsymbol{w}^{\star\top}_\tau \bar{\boldsymbol{h}}_t + b^\star_\tau)$
  $\nu_t \leftarrow \nu_{\max}\, \sigma(\boldsymbol{w}^{\star\top}_\nu \bar{\boldsymbol{h}}_t + b^\star_\nu)$
  $\boldsymbol{d}^{\text{lstm,n}}_t \leftarrow \frac{\|\boldsymbol{g}_t\|}{\|\boldsymbol{d}^{\text{lstm}}_t\| + \varepsilon_{\text{dir}}}\, \boldsymbol{d}^{\text{lstm}}_t$
  $\tilde{\boldsymbol{d}}_t \leftarrow -\tau_t\, \boldsymbol{g}_t + \nu_t\, \boldsymbol{d}^{\text{lstm,n}}_t$
  $\boldsymbol{d}_t \leftarrow \tilde{\boldsymbol{d}}_t - \frac{\max\{0,\, \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa\|\boldsymbol{g}_t\|^2\}}{\|\boldsymbol{g}_t\|^2}\, \boldsymbol{g}_t$
  // — Armijo line search —
  $\eta_t \leftarrow 1$
  while $f(\boldsymbol{x}_t + \eta_t \boldsymbol{d}_t) > f(\boldsymbol{x}_t) + c_1 \eta_t \boldsymbol{g}_t^\top \boldsymbol{d}_t$ **do**
    $\eta_t \leftarrow \beta\, \eta_t$
  end while
  $\hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t + \eta_t\, \boldsymbol{d}_t$
  if $\|\nabla f(\hat{\boldsymbol{x}}_t)\| < \varepsilon_{\text{tol}}$ **then**
    $\boldsymbol{x}_{t+1} \leftarrow \hat{\boldsymbol{x}}_t$; **return** $\boldsymbol{x}_{t+1}$ *// Stage 2 skipped; $\hat{\boldsymbol{x}}_t$ becomes the next iterate*
  end if
  // — Stage 2: Learnable L-BFGS candidate —
  $\mathcal{M}_0 \leftarrow \{(\boldsymbol{s}^{\text{lstm}}_h, \boldsymbol{y}^{\text{lstm}}_h) : H - m \le h \le H - 1,\ \boldsymbol{y}^{\text{lstm}\top}_h \boldsymbol{s}^{\text{lstm}}_h > \varepsilon_{\text{curv}}\}$
  $\boldsymbol{z}_t \leftarrow \text{LearnableLBFGS}(\hat{\boldsymbol{x}}_t, K, \mathcal{M}_0, m, \varepsilon_{\text{tol}}, \boldsymbol{v}^\star, \tilde{\boldsymbol{\mu}}^\star)$
  // — Stage 2: Accept or reject the refinement endpoint —
  if $f(\boldsymbol{z}_t) \le f(\hat{\boldsymbol{x}}_t)$ **then**
    $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{z}_t$
  else
    $\boldsymbol{x}_{t+1} \leftarrow \hat{\boldsymbol{x}}_t$
  end if
end for
return $\boldsymbol{x}_{T^{\text{test}}_{\max}}$ *// last accepted iterate*

- Total: $O\big(n(HD^2 + mK)\big)$ plus $O(n)$ per Armijo trial, excluding the evaluations of f and ∇f, and a negligible overhead for the final acceptance test.
- Oracle calls: gradients at no more than $H + K$ new points (the $H - 1$ trial points after $\boldsymbol{x}_t$, the point $\hat{\boldsymbol{x}}_t$, and at most K refinement points; the gradient of $\boldsymbol{x}_{t+1}$ is among them), and objective values at every Armijo trial point and at $\boldsymbol{z}_t$.

G Technical Proofs

G.1 Proof of Lemma 1

Verification of the projection formula. Write $\boldsymbol{g} = \boldsymbol{g}_t$, $\tilde{\boldsymbol{d}} = \tilde{\boldsymbol{d}}_t$. The half-space $\mathcal{H}_t$ is $\{\boldsymbol{d} : \boldsymbol{g}^\top \boldsymbol{d} \le -\kappa\|\boldsymbol{g}\|^2\}$. If $\tilde{\boldsymbol{d}} \in \mathcal{H}_t$ the max evaluates to 0 and $\boldsymbol{d}_t = \tilde{\boldsymbol{d}}_t$. Otherwise, the projection onto the boundary hyperplane $\boldsymbol{g}^\top \boldsymbol{d} = -\kappa\|\boldsymbol{g}\|^2$ along the normal $\boldsymbol{g}$ gives

$$\boldsymbol{d}_t = \tilde{\boldsymbol{d}} - \frac{\boldsymbol{g}^\top \tilde{\boldsymbol{d}} + \kappa\|\boldsymbol{g}\|^2}{\|\boldsymbol{g}\|^2}\, \boldsymbol{g}.$$

Then $\boldsymbol{g}^\top \boldsymbol{d}_t = \boldsymbol{g}^\top \tilde{\boldsymbol{d}} - (\boldsymbol{g}^\top \tilde{\boldsymbol{d}} + \kappa\|\boldsymbol{g}\|^2) = -\kappa\|\boldsymbol{g}\|^2$, confirming the sufficient-descent condition. This point is also the closest point of $\mathcal{H}_t$ to $\tilde{\boldsymbol{d}}$: for any $\boldsymbol{d} \in \mathcal{H}_t$, $\boldsymbol{g}^\top(\tilde{\boldsymbol{d}} - \boldsymbol{d}) \geq \boldsymbol{g}^\top \tilde{\boldsymbol{d}} + \kappa\|\boldsymbol{g}\|^2 > 0$, so by Cauchy–Schwarz $\|\tilde{\boldsymbol{d}} - \boldsymbol{d}\| \geq (\boldsymbol{g}^\top \tilde{\boldsymbol{d}} + \kappa\|\boldsymbol{g}\|^2)/\|\boldsymbol{g}\| = \|\tilde{\boldsymbol{d}} - \boldsymbol{d}_t\|$. □

Remark 1 (Descent guarantee)**.** *By construction, $\boldsymbol{g}_t^\top \boldsymbol{d}_t \leq -\kappa\,\|\boldsymbol{g}_t\|^2$, so $\boldsymbol{d}_t$ is always a sufficient descent direction whenever $\boldsymbol{g}_t \neq \boldsymbol{0}$. Moreover, as shown in the proof above, $\boldsymbol{d}_t$ is the closest vector in $\mathcal{H}_t$ to $\tilde{\boldsymbol{d}}_t$ in Euclidean norm. We test the sufficient-descent condition $\boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t \leq -\kappa\|\boldsymbol{g}_t\|^2$ first to determine whether the projection in (8) is needed.*

G.2 Proof of Proposition 1

Proof. If the algorithm terminates finitely, then it does so only when $\boldsymbol{g}_t = \nabla f(\boldsymbol{x}_t) = \boldsymbol{0}$, in which case $\boldsymbol{x}_t$ is stationary. Hence it remains to consider the case of an infinite sequence. Fix any iteration t. We prove that

$$\boldsymbol{g}_t^\top \boldsymbol{d}_t \leq -\kappa\|\boldsymbol{g}_t\|^2, \tag{28}$$

$$\|\boldsymbol{d}_t\| \leq C_d\|\boldsymbol{g}_t\|, \tag{29}$$

$$\eta_t \geq \eta_{\min}, \tag{30}$$

$$f(\boldsymbol{x}_{t+1}) \leq f(\boldsymbol{x}_t) - c_1\eta_{\min}\kappa\|\boldsymbol{g}_t\|^2, \tag{31}$$

for

$$C_d := 2(\tau_{\max} + \nu_{\max}) + \kappa \qquad \text{and} \qquad \eta_{\min} := \min\left\{1,\ \beta\frac{2(1-c_1)\kappa}{LC_d^2}\right\} > 0.$$

Consequently,

$$\sum_{t=0}^{\infty} \|\boldsymbol{g}_t\|^2 < \infty. \tag{32}$$

Step 1: We bound the normalized learned direction. From the definition of $\boldsymbol{d}_t^{\mathrm{lstm,n}}$, we have

$$\|\boldsymbol{d}_t^{\mathrm{lstm,n}}\| = \frac{\|\boldsymbol{g}_t\|\,\|\boldsymbol{d}_t^{\mathrm{lstm}}\|}{\|\boldsymbol{d}_t^{\mathrm{lstm}}\| + \varepsilon_{\mathrm{dir}}} \leq \|\boldsymbol{g}_t\|.$$

Hence, by (A3),

$$\|\tilde{\boldsymbol{d}}_t\| \leq \tau_t\|\boldsymbol{g}_t\| + \nu_t\|\boldsymbol{d}_t^{\mathrm{lstm,n}}\| \leq (\tau_{\max} + \nu_{\max})\|\boldsymbol{g}_t\|. \tag{33}$$

Step 2: We derive the sufficient-descent property. Define

$$\alpha_t := \frac{\max\{0,\ \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa\|\boldsymbol{g}_t\|^2\}}{\|\boldsymbol{g}_t\|^2}.$$

Then (8) becomes

$$\boldsymbol{d}_t = \tilde{\boldsymbol{d}}_t - \alpha_t\boldsymbol{g}_t.$$

Therefore

$$\boldsymbol{g}_t^\top \boldsymbol{d}_t = \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t - \alpha_t\|\boldsymbol{g}_t\|^2.$$

If $\boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa\|\boldsymbol{g}_t\|^2 \leq 0$, then $\alpha_t = 0$ and

$$\boldsymbol{g}_t^\top \boldsymbol{d}_t = \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t \leq -\kappa\|\boldsymbol{g}_t\|^2.$$

Otherwise,

$$\alpha_t = \frac{\boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa\|\boldsymbol{g}_t\|^2}{\|\boldsymbol{g}_t\|^2},$$

and hence

$$\boldsymbol{g}_t^\top \boldsymbol{d}_t = \boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t - \left(\boldsymbol{g}_t^\top \tilde{\boldsymbol{d}}_t + \kappa\|\boldsymbol{g}_t\|^2\right) = -\kappa\|\boldsymbol{g}_t\|^2.$$

Thus (28) holds in all cases.

We next bound the projected direction. From the definition of α_t and Cauchy–Schwarz,

$$0 \leq \alpha_t \leq \frac{\|\boldsymbol{g}_t\|\,\|\tilde{\boldsymbol{d}}_t\| + \kappa\|\boldsymbol{g}_t\|^2}{\|\boldsymbol{g}_t\|^2} = \frac{\|\tilde{\boldsymbol{d}}_t\|}{\|\boldsymbol{g}_t\|} + \kappa.$$

Using (33),

$$\alpha_t \le \tau_{\max} + \nu_{\max} + \kappa.$$

Therefore

$$\begin{aligned}\|\boldsymbol{d}_t\| &\le \|\tilde{\boldsymbol{d}}_t\| + \alpha_t\|\boldsymbol{g}_t\| \\ &\le (\tau_{\max} + \nu_{\max})\|\boldsymbol{g}_t\| + (\tau_{\max} + \nu_{\max} + \kappa)\|\boldsymbol{g}_t\| \\ &= \big(2(\tau_{\max} + \nu_{\max}) + \kappa\big)\|\boldsymbol{g}_t\| = C_d\|\boldsymbol{g}_t\|.\end{aligned}$$

This proves (29).

Step 3: We obtain finite termination of Armijo backtracking and a uniform lower bound on η_t. Since ∇f is globally Lipschitz continuous with constant L by (A2), the descent lemma gives, for every $\eta \ge 0$,

$$f(\boldsymbol{x}_t + \eta\boldsymbol{d}_t) \le f(\boldsymbol{x}_t) + \eta\boldsymbol{g}_t^\top\boldsymbol{d}_t + \frac{L}{2}\eta^2\|\boldsymbol{d}_t\|^2.$$

Using (28) and (29), it follows that

$$f(\boldsymbol{x}_t + \eta\boldsymbol{d}_t) \le f(\boldsymbol{x}_t) + \eta\boldsymbol{g}_t^\top\boldsymbol{d}_t + \frac{L}{2}\eta^2 C_d^2\|\boldsymbol{g}_t\|^2.$$

Therefore Armijo (9) is guaranteed whenever

$$\frac{L}{2}\eta^2 C_d^2\|\boldsymbol{g}_t\|^2 \le (1-c_1)\eta\big(-\boldsymbol{g}_t^\top\boldsymbol{d}_t\big).$$

Using (28), it suffices that

$$\frac{L}{2}\eta C_d^2 \le (1-c_1)\kappa,$$

i.e.,

$$\eta \le \bar{\eta} := \frac{2(1-c_1)\kappa}{LC_d^2}.$$

Hence every $\eta \in (0, \bar{\eta}]$ satisfies Armijo, so backtracking terminates finitely.

Now consider the discrete backtracking sequence $\{1, \beta, \beta^2, \dots\}$. If $\bar{\eta} \ge 1$, then $\eta_t = 1$ is accepted. If $\bar{\eta} < 1$, backtracking accepts no later than the first power β^J satisfying $\beta^J \le \bar{\eta}$ (the exponent J is local to this argument and unrelated to the memory capacity m), so

$$\eta_t \ge \beta^J > \beta\bar{\eta}.$$

Thus, in all cases,

$$\eta_t \ge \eta_{\min} := \min\left\{1,\ \beta\frac{2(1-c_1)\kappa}{LC_d^2}\right\},$$

which proves (30).

By Armijo (9), (28), and (30),

$$f(\boldsymbol{x}_{t+1}) \le f(\boldsymbol{x}_t) + c_1\eta_t\boldsymbol{g}_t^\top\boldsymbol{d}_t \le f(\boldsymbol{x}_t) - c_1\eta_t\kappa\|\boldsymbol{g}_t\|^2 \le f(\boldsymbol{x}_t) - c_1\eta_{\min}\kappa\|\boldsymbol{g}_t\|^2.$$

This proves (31). In particular, $\{f(\boldsymbol{x}_t)\}$ is monotone decreasing, so every iterate remains in $\mathcal{X}_0$.

Step 4: We prove that the gradients vanish. Let

$$f_{\inf} := \inf_{\boldsymbol{x}\in\mathcal{X}_0} f(\boldsymbol{x}) > -\infty$$

from (A1). Summing (31) from $t = 0$ to T gives

$$c_1\eta_{\min}\kappa\sum_{t=0}^{T}\|\boldsymbol{g}_t\|^2 \le f(\boldsymbol{x}_0) - f(\boldsymbol{x}_{T+1}) \le f(\boldsymbol{x}_0) - f_{\inf}.$$

Letting $T \to \infty$ yields (32). Since the summands are nonnegative, (32) implies

$$\|\boldsymbol{g}_t\| \to 0,$$

which proves (17).

Finally, if $\mathcal{X}_0$ is compact, then $\{\boldsymbol{x}_t\} \subset \mathcal{X}_0$ has accumulation points. Let $\boldsymbol{x}_\infty$ be any accumulation point, with $\boldsymbol{x}_{t_j} \to \boldsymbol{x}_\infty$. By continuity of ∇f and the full limit $\|\boldsymbol{g}_t\| \to 0$ established above,

$$\nabla f(\boldsymbol{x}_\infty) = \lim_{j\to\infty}\nabla f(\boldsymbol{x}_{t_j}) = \boldsymbol{0}.$$

Thus every accumulation point is stationary. □

G.3 Proof of Proposition 2

Proof. If the algorithm terminates finitely, then it does so only when $\boldsymbol{g}_t = \nabla f(\boldsymbol{x}_t) = \mathbf{0}$, in which case $\boldsymbol{x}_t$ is stationary. Hence it remains to consider the case of an infinite sequence.

We prove that the following hold for every t:

$$f(\boldsymbol{x}_{t+1}) \le f(\hat{\boldsymbol{x}}_t), \tag{34}$$

$$f(\boldsymbol{x}_{t+1}) \le f(\boldsymbol{x}_t) - c_1 \eta_{\min} \kappa \|\boldsymbol{g}_t\|^2, \tag{35}$$

$$\boldsymbol{x}_t \in \mathcal{X}_0, \tag{36}$$

where $\mathcal{X}_0$ is the level set of (A1), with the constant

$$\eta_{\min} := \min\left\{1,\ \beta \frac{2(1-c_1)\kappa}{L C_d^2}\right\} > 0$$

from the proof of Proposition 1. Consequently,

$$\sum_{t=0}^{\infty} \|\boldsymbol{g}_t\|^2 < \infty. \tag{37}$$

By Steps 1–3 of the proof of Proposition 1, which apply at any nonstationary current point independently of the preceding trajectory (there the accepted iterate is the Stage-1 point itself, $\boldsymbol{x}_{t+1} = \hat{\boldsymbol{x}}_t$, because Stage 2 is disabled), the safeguarded Stage-1 step satisfies the uniform decrease estimate

$$f(\hat{\boldsymbol{x}}_t) \le f(\boldsymbol{x}_t) - c_1 \eta_{\min} \kappa \|\boldsymbol{g}_t\|^2. \tag{38}$$

By the acceptance test (13), either $\boldsymbol{x}_{t+1} = \boldsymbol{z}_t$ with $f(\boldsymbol{z}_t) \le f(\hat{\boldsymbol{x}}_t)$, or $\boldsymbol{x}_{t+1} = \hat{\boldsymbol{x}}_t$. In both cases,

$$f(\boldsymbol{x}_{t+1}) \le f(\hat{\boldsymbol{x}}_t),$$

which proves (34). Combining this with (38) gives

$$f(\boldsymbol{x}_{t+1}) \le f(\boldsymbol{x}_t) - c_1 \eta_{\min} \kappa \|\boldsymbol{g}_t\|^2,$$

which proves (35).

We now prove (36) by induction. Clearly $\boldsymbol{x}_0 \in \mathcal{X}_0$. Assume $\boldsymbol{x}_t \in \mathcal{X}_0$. Then (35) implies

$$f(\boldsymbol{x}_{t+1}) \le f(\boldsymbol{x}_t) \le f(\boldsymbol{x}_0),$$

hence $\boldsymbol{x}_{t+1} \in \mathcal{X}_0$. Therefore every iterate belongs to $\mathcal{X}_0$.

Because f is bounded below on $\mathcal{X}_0$ by (A1), letting

$$f_{\inf} := \inf_{\boldsymbol{x} \in \mathcal{X}_0} f(\boldsymbol{x}) > -\infty,$$

and summing (35) from $t = 0$ to T yields

$$c_1 \eta_{\min} \kappa \sum_{t=0}^{T} \|\boldsymbol{g}_t\|^2 \le f(\boldsymbol{x}_0) - f(\boldsymbol{x}_{T+1}) \le f(\boldsymbol{x}_0) - f_{\inf}.$$

Letting $T \to \infty$ proves (37).

Since the summands are nonnegative, (37) implies

$$\|\boldsymbol{g}_t\| \to 0,$$

which proves (18).

Finally, if $\mathcal{X}_0$ is compact, then the sequence $\{\boldsymbol{x}_t\}$ has at least one accumulation point. Let $\boldsymbol{x}_\infty$ be any accumulation point, and let $\boldsymbol{x}_{t_j} \to \boldsymbol{x}_\infty$ along a subsequence. By continuity of ∇f and the full limit $\|\boldsymbol{g}_t\| \to 0$ established above,

$$\nabla f(\boldsymbol{x}_\infty) = \lim_{j \to \infty} \nabla f(\boldsymbol{x}_{t_j}) = \mathbf{0}.$$

Thus every accumulation point is stationary. □

H Ablation Study

We ablate the two stages of MINN on least squares: the LSTM direction (Stage 1) and the L-BFGS refinement (Stage 2), and, within Stage 2, whether its pair weights v and step sizes μ are learned. All variants share the initialization, training set, minibatch order, and test protocol, with stopping criterion $\|\nabla f\| < 10^{-2}$. Models are trained at $n = 200$ and tested on 100 instances for each $n \in \{200, \dots, 1000\}$ with in-distribution least-squares instances.

Two-phase evaluation protocol. The *phases* here describe the deployment schedule, whereas Stage 1 and Stage 2 are the operations within each MINN outer iteration. In Phase 1 (the learned phase), each variant runs up to $T_{\text{warm}} = 3$ outer iterations, with $K = 25$ refinement steps per iteration when Stage 2 is present. If the stopping criterion has not been reached, Phase 2 (classical continuation) starts standard L-BFGS from the final Phase-1 iterate, seeded with its retained curvature memory $\mathcal{M}_{\text{handoff}}$, and continues toward the same gradient tolerance. For variants with Stage 2, this is the memory left by the last refinement; without Stage 2, it contains the final LSTM trial pairs. Retaining memory at this handoff is specific to this evaluation protocol: Algorithm 4 instead repeats the safeguarded two-stage iteration and discards refinement memory after each call. Reported solve times include both phases, so the table compares complete deployments rather than Stage 2 in isolation. The untrained L-BFGS reference has no learned phase.

Ablation variants. Each variant changes exactly one component and leaves the rest unchanged: the initialization, the training data and minibatch order, the meta-training objective, T_{warm}, and the Phase-2 continuation rule.

w/o LSTM. Stage 1 is the identity, $\hat{x}_t = x_t$. Since there are no LSTM curvature pairs, the refinement memory starts empty and is *not* reset between outer iterations. Phase 1 therefore runs up to $T_{\text{warm}}K$ consecutive iterations of (12) with a single persistent memory. Only v, μ are trained; the LSTM parameters are frozen and unused.

w/o L-BFGS. Stage 2 is the identity, $x_{t+1} = \hat{x}_t$ (i.e. $K = 0$). The memory is still reset to the LSTM curvature pairs at every outer step, and Phase 2 is seeded with them. Only the LSTM is trained; v, μ are frozen and unused.

Fixed L-BFGS. Stage 2 is kept, but with $v_i \equiv 1$ and $\mu_k \equiv 1$ for $k \geq 2$, and nothing in it is learned for L-BFGS. Equations (11) and (12) then reduce to the standard two-loop recursion ($\rho_i = 1/(s_i^\top y_i)$) with a unit step and no line search. Only the LSTM is trained.

Table 4: Component ablation over $n \in \{200, \dots, 1000\}$. Time: seconds on an NVIDIA H100 GPU for the complete two-phase solve to reach $\|\nabla f\| < 10^{-2}$ (median over all instances). Ratio: geometric mean over dimensions of the time ratio to full MINN (lower is better).

Variant	LSTM	L-BFGS	Time (s)	Ratio
MINN (full)	✓	learned	0.712	1
w/o LSTM	–	learned	0.853	1.20
w/o L-BFGS	✓	–	0.885	1.24
fixed L-BFGS (v=1, μ=1)	✓	fixed	0.911	1.28
L-BFGS (untrained)	–	–	2.577	3.62

Full MINN is the fastest variant. It reaches $\|\nabla f\| < 10^{-2}$ in 0.712 s, and every ablated variant is slower. Each component contributes: removing the LSTM makes the solve 20% slower, removing the L-BFGS refinement 24% slower, and fixing the L-BFGS parameters 28% slower. The LSTM has the smallest effect.

Stage 2 helps only when its parameters are learned. With fixed parameters, the solve takes 0.911 s, which is slower than having no L-BFGS refinement at all (0.885 s).

No single component accounts for the gain over standard L-BFGS. Full MINN is 3.6× faster than untrained L-BFGS (0.712 s vs. 2.577 s), and each ablated variant is still 2.8–3.0× faster than it.

I TRAINING COST

Table 5 reports the offline training time of each learned optimizer, measured on the system described in Section E (an NVIDIA H100 GPU and an AMD EPYC 9634 CPU). For each task, the ID and OOD columns report the recorded training times for the in-distribution and out-of-distribution settings. The classical baselines (L-BFGS, Adam, and Nesterov) require no offline training and are omitted. The learned baselines are trained on the same data as MINN (Section 5) with their default settings (Section E), so each reported time reflects that method's own training schedule.

Table 5: Offline training time (seconds) of the learned optimizers. ID and OOD denote the recorded training times for the in-distribution and out-of-distribution settings. Least-squares and Rastrigin optimizers are trained at n=200. Parentheses give the ratio to MINN's time in the same column. Lowest time in **bold**.

	Least squares		Rastrigin		MNIST MLP	
Method	ID	OOD	ID	OOD	ID	OOD
cLSTM	121.3 (2.0×)	124.8 (2.4×)	146.9 (1.9×)	165.6 (1.6×)	653.2 (16.4×)	989.4 (21.0×)
RNNProp	1281.7 (21.3×)	1625.4 (31.0×)	1719.2 (22.7×)	1506.4 (14.5×)	924.9 (23.2×)	991.5 (21.0×)
L2O-DM	2229.5 (37.0×)	1765.1 (33.7×)	2247.6 (29.7×)	2129.0 (20.5×)	977.2 (24.5×)	1013.2 (21.5×)
MINN	**60.2**	**52.4**	**75.8**	**104.1**	**39.9**	**47.2**

MINN is the cheapest learned optimizer to train. In every setting the training times are ordered MINN $<$ cLSTM $<$ RNNProp $<$ L2O-DM. MINN trains in 39.9–104.1 s, under two minutes in every setting, whereas cLSTM takes 1.6–21.0× as long, RNNProp 14.5–31.0×, and L2O-DM 20.5–37.0×. Summed over the six settings, MINN needs 379.6 s, compared with 2201.2 s for cLSTM, 8049.1 s for RNNProp, and 10361.6 s for L2O-DM (5.8×, 21.2×, and 27.3× MINN's total).

The gap depends on the task. On the two synthetic tasks, cLSTM costs only 1.6–2.4× as much as MINN, while RNNProp and L2O-DM cost 14.5–37.0× as much. On MNIST, cLSTM's time rises to 653.2–989.4 s, and all three baselines take 16.4–24.5× as long as MINN. MINN's own training time is lowest on MNIST (39.9 s and 47.2 s), although the MLP has n=15,910 parameters and the synthetic training instances have n=200. The tasks also differ in objective, training data, and when training stops, so this comparison does not isolate the effect of dimension; it shows only that MINN's training cost stayed low on the highest-dimensional task.

Lower training cost comes with better solution quality. The baselines' higher training cost does not buy robustness. They are faster per test instance (0.09–0.31 s, versus 0.51–4.39 s for MINN; Tables 6 to 8, 10 and 11), but they stall or diverge at larger dimensions and under distribution shift. In every test setting of these tables, MINN costs less to train and attains a lower mean best objective value and gradient norm than every learned baseline. Table 5 does not include the least-squares models trained at n=400 (Table 9).

Amortization relative to L-BFGS. The introduction asks whether repeated savings compensate for offline training. Multiplying the difference in mean time at termination by the number of test instances at each test dimension and summing over dimensions gives the total time MINN saves over L-BFGS in each setting. One training run serves all test dimensions of a task, since the learned optimizers are tested at larger dimensions without retraining (Section 5). On least squares, MINN saves 2.1×10^3 s (ID, 35× its training time) and 1.5×10^4 s (OOD, 291×). On MNIST the saving per initialization is small (0.79 s ID, 0.36 s OOD), so MINN's training time is recovered only after about 51 (ID) and 131 (OOD) solves, compared with the 50 test initializations. On Rastrigin (ID), the same calculation gives 2.6×10^4 s over the 500 test instances (342×), but this figure mostly reflects L-BFGS's stopping rule: L-BFGS reaches objective values similar to MINN's within about the first second but runs much longer before stopping (Figure 6). Measured by time to a comparable objective value, the saving on Rastrigin would be far smaller and may be negligible. All these estimates use times at termination, so they measure savings under the reported stopping rules, not the time to reach a common accuracy, and none should be read as an exact break-even point.

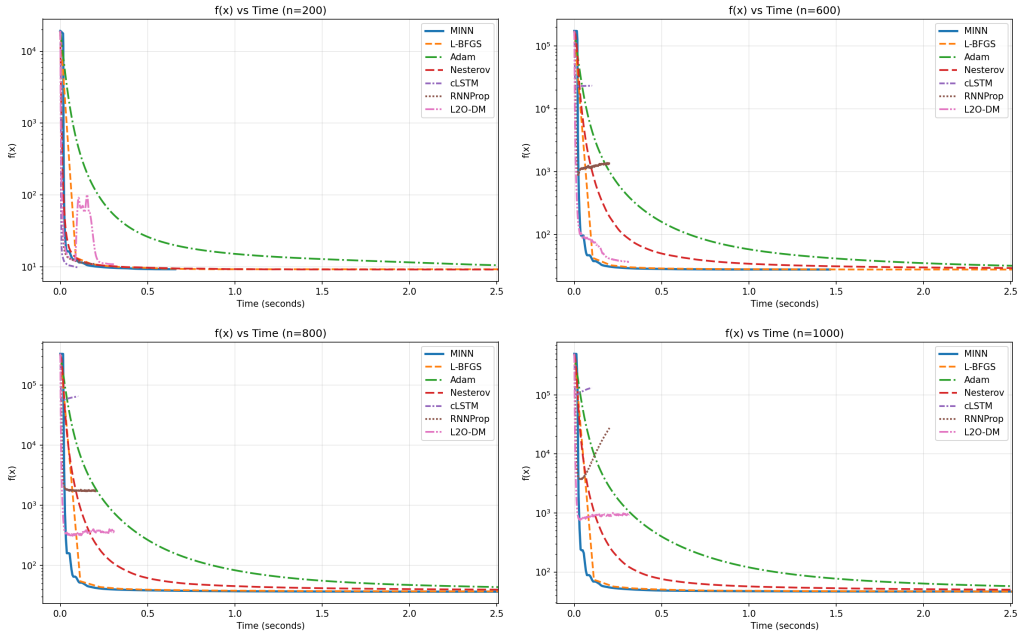

Figure 6: Non-convex Rastrigin: convergence of $f(\boldsymbol{x}_t)$ vs. wall-clock time for $n = 200$, 400, 800, and 1000. MINN reaches objective values comparable to or better than L-BFGS and terminates 17–67× sooner (Table 6); purely learned baselines stall or diverge at larger dimensions.

Table 6: Non-convex Rastrigin: mean best $f(\boldsymbol{x})$, $\|\nabla f(\boldsymbol{x})\|$, and runtime across 100 test instances per dimension. All learned optimizers are trained at n=200. Best learned method in **bold**; † denotes best overall.

	n=200			n=400			n=600			n=800			n=1000		
Method	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time
L-BFGS	9.21e0	1.75e-2	44.43	1.91e1	4.36e-2	58.27	2.78e1	6.20e-2	52.40	3.69e1	8.51e-2	57.30	4.64e1	1.07e-1	56.07
Adam	9.14e0	1.62e-1	4.68	1.93e1	2.27e0	4.81	2.84e1	3.57e0	4.74	3.83e1	5.32e0	4.74	4.85e1	9.71e0	4.74
Nesterov	9.14e0†	2.88e-2	3.13	1.91e1	3.51e-2	3.26	2.88e1	1.16e0	3.29	3.89e1	1.22e0	3.29	4.88e1	1.25e0	3.29
cLSTM	9.77e0	2.53e-1	0.10	1.98e1	5.27e0	0.10	2.10e4	9.44e3	0.10	5.45e4	1.72e4	0.10	9.64e4	2.50e4	0.10
RNNProp	1.07e1	1.85e1	0.20	3.42e2	6.76e2	0.20	8.07e2	1.42e3	0.20	1.50e3	2.45e3	0.20	3.55e3	4.31e3	0.20
L2O-DM	1.07e1	9.17e0	0.31	2.18e1	2.59e1	0.31	3.65e1	4.92e1	0.31	2.44e2	4.59e2	0.31	6.78e2	9.47e2	0.31
MINN	**9.23e0**	**1.72e-2**†	**0.66**	**1.91e1**†	**1.63e-2**†	**1.06**	**2.78e1**†	**1.80e-2**†	**1.49**	**3.65e1**†	**1.31e-2**†	**2.72**	**4.64e1**†	**1.48e-2**†	**3.19**

J Additional Experiments

This section reports the non-convex Rastrigin experiments, additional least-squares results for training at n=200 and at n=400, and additional MNIST MLP results, followed by a summary of findings.

J.1 Experiment 3: Non-Convex Rastrigin Functions

We evaluate on the generalized Rastrigin family, which features a highly multi-modal landscape with many local minima. All learned optimizers are trained at n=200 and tested at $n \in \{200, 400, 600, 800, 1000\}$. Figure 6 shows convergence trajectories and Table 6 reports summary statistics.

Observations. On the non-convex Rastrigin benchmark, MINN matches or improves upon the L-BFGS objective for $n \geq 400$ and is marginally higher at n=200 (9.23 vs. 9.21), while terminating **17–67× sooner** (e.g., 0.66 s vs. 44.4 s at n=200; 3.19 s vs. 56.1 s at n=1000). These ratios compare termination times; Figure 6 shows that L-BFGS reaches similar objective values within the first second but runs much longer before stopping. Crucially, MINN achieves gradient norms of order 10^{-2}—comparable to or better than L-BFGS—indicating approximate first-order stationarity; gradient norms alone do not distinguish local minima from saddle points. Among classical methods, Nesterov is fast

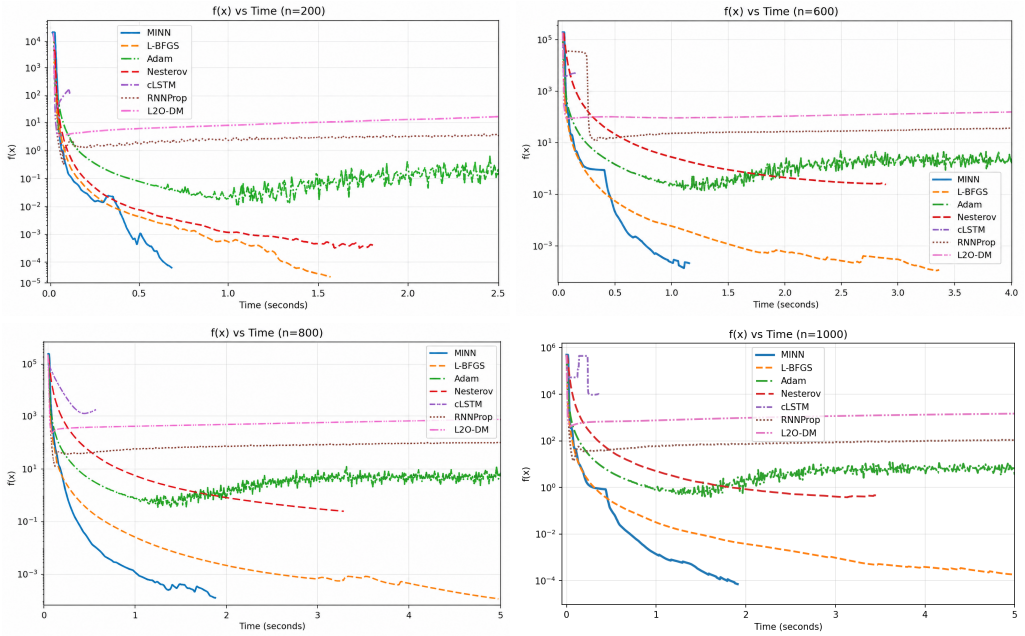

Figure 7: Random least-squares (in-distribution): convergence of $f(\boldsymbol{x}_t)$ vs. wall-clock time.

Table 7: Random least-squares (in-distribution): mean best $f(\boldsymbol{x})$, $\|\nabla f(\boldsymbol{x})\|$, and runtime (seconds) across 100 test instances per dimension. All learned optimizers are trained at n=200 and tested at dimensions up to n=1000. Best learned method in **bold**; † denotes best overall.

	n=200			n=400			n=600			n=800			n=1000		
Method	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time
L-BFGS	3.03e-4	9.20e-3	1.23	9.90e-5†	9.49e-3	1.74	1.62e-4†	9.49e-3†	2.61	1.64e-4	9.68e-3†	3.49	9.74e-5	9.68e-3†	4.39
Adam	1.12e-3	5.78e-1	5.13	1.07e-2	2.92e0	4.89	2.13e-2	4.88e0	4.93	6.11e-2	9.90e0	4.91	9.00e-2	1.25e1	5.08
Nesterov	4.80e-4	1.08e-2	1.79	3.85e-4	1.21e-2	2.03	2.67e-1	2.68e0	3.16	3.33e-1	2.91e0	3.14	3.52e-1	2.96e0	3.23
cLSTM	4.33e1	5.88e1	0.13	1.30e3	1.02e3	0.12	3.43e3	2.03e3	0.12	6.01e3	3.13e3	0.15	9.43e3	4.34e3	0.15
RNNProp	2.78e-1	4.37e0	6.25	2.01e0	1.77e1	6.01	4.79e0	3.17e1	6.08	7.98e0	5.19e1	6.03	1.17e1	7.38e1	6.21
L2O-DM	3.34e0	3.74e1	6.22	1.76e1	1.38e2	6.00	7.71e1	3.18e2	6.02	2.14e2	5.58e2	6.02	4.21e2	8.39e2	6.20
MINN	**2.83e-4**†	**8.96e-3**†	**0.52**	**1.05e-4**	**9.24e-3**†	**0.77**	**2.80e-4**	**9.50e-3**	**0.99**	**1.33e-4**†	**9.70e-3**	**1.38**	**7.81e-5**†	**9.76e-3**	**1.58**

but stagnates at suboptimal gradient norms ($\sim$1.0) for $n \geq 600$, while Adam converges slowly and, for $n \geq 400$, to inferior solutions. Purely learned methods again diverge catastrophically for $n \geq 600$ (cLSTM reaches 10^4–10^5), while MINN's safeguards ensure reliable convergence.

J.2 Experiment 1: Least-Squares

J.2.1 Training with $n = 200$

In-Distribution Setting. All learned methods are trained on instances with $A_{ij} \sim \mathcal{N}(0, 1)$ at n=200 and tested on the same distribution at $n \in \{200, \ldots, 1000\}$. Figure 7 shows convergence trajectories and Table 7 reports summary statistics.

Figure 7 reveals three consistent trends. First, in the matched train–test setting at n=200, the purely learned optimizers (cLSTM, RNNProp, and L2O-DM) make rapid initial progress, reflecting the benefit of amortization. However, their progress quickly saturates at low or moderate accuracy, and the trajectories fail to enter the high-accuracy regime. This behavior is consistent with prior observations on learned optimizers (Gao et al., 2024; Chen et al., 2022): they can accelerate the early phase of optimization but often lack the numerical mechanisms needed for reliable final convergence. The learned baselines become unstable or diverge for $n \geq 400$—their mean best objective values range from 6.3 (L2O-DM, n=400) to 1.2×10^5 (cLSTM, n=1000)—confirming the generalization brittleness identified in Section 1. MINN's classical safeguards prevent such divergence entirely.

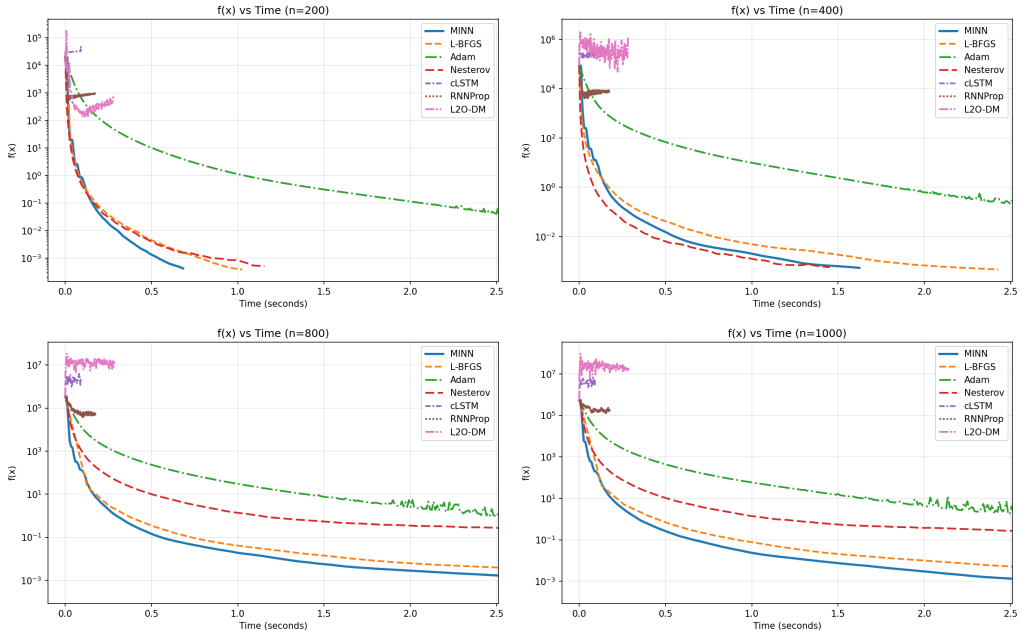

Figure 8: Random least-squares (OOD): convergence of $f(\boldsymbol{x}_t)$ vs. wall-clock time.

Table 8: Random least-squares (OOD): mean best $f(\boldsymbol{x})$, $\|\nabla f(\boldsymbol{x})\|$, and runtime across 100 test instances. Training at n=200 with $\mathcal{N}(0,1)$; testing with $\mathcal{N}(0.15,1)$ at larger dimensions. Best learned method in **bold**; † denotes best overall.

	n=200			n=400			n=600			n=800			n=1000		
Method	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time
L-BFGS	2.12e-4	9.32e-3	1.03	2.67e-4†	9.11e-3†	2.43	2.56e-4	1.41e-2	33.42	7.45e-4	3.26e-2	63.62	8.27e-4	5.37e-2	65.06
Adam	3.47e-3	9.70e-3	3.62	2.13e-2	3.50e-1	5.06	6.09e-2	9.22e-1	5.13	1.16e-1	1.54e0	5.10	2.19e-1	3.56e0	5.13
Nesterov	4.34e-4	1.05e-2	1.16	4.58e-4	1.06e-2	1.45	1.37e-1	1.56e0	3.09	2.07e-1	1.60e0	3.09	2.08e-1	1.60e0	3.10
cLSTM	5.73e3	3.18e3	0.09	1.85e4	8.76e3	0.09	2.08e4	1.09e4	0.09	4.27e4	2.17e4	0.09	5.43e4	2.57e4	0.09
RNNProp	3.36e2	5.96e2	0.17	1.23e3	1.00e3	0.17	7.58e3	2.99e3	0.18	1.38e4	6.20e3	0.18	6.54e4	1.85e4	0.18
L2O-DM	2.46e1	5.31e1	0.28	3.31e3	1.69e3	0.28	2.01e4	1.39e4	0.29	8.96e4	3.19e4	0.28	1.27e5	3.85e4	0.29
MINN	**1.39e-4**†	**9.21e-3**†	**0.68**	**2.91e-4**	**9.11e-3**	**1.63**	**2.17e-4**†	**9.37e-3**†	**2.36**	**2.02e-4**†	**9.52e-3**†	**4.20**	**2.50e-4**†	**9.63e-3**†	**4.39**

Second, the classical baselines exhibit the opposite behavior. L-BFGS, Adam, and Nesterov are generally slower during the earliest iterations, but their trajectories continue to decrease after the learned methods have stalled. In particular, L-BFGS and Nesterov reach substantially higher solution quality than the purely learned baselines, illustrating the value of classical descent and curvature mechanisms for high-accuracy optimization.

Third, MINN combines the advantages of both regimes. Its trajectory decreases as rapidly as learned methods in the initial phase, but continues to improve like a classical method in the high-accuracy regime. Across $n \in \{200, 400, 600, 800, 1000\}$, MINN remains among the fastest methods to reach low objective values and closely tracks, or improves upon, the best classical baseline. The gap becomes more pronounced as dimension increases: while the purely learned methods become unstable or diverge for $n > 400$, MINN remains stable and convergent. This supports the central design principle of MINN: the coordinate-wise LSTM supplies amortized, dimension-agnostic search directions, whereas the learnable L-BFGS stage supplies curvature adaptation and the classical safeguards—half-space projection, Armijo backtracking, and the endpoint acceptance test—supply numerical stabilization.

Out-of-Distribution Setting. We shift the test distribution to $A_{ij} \sim \mathcal{N}(0.15, 1)$ while keeping the training distribution at $\mathcal{N}(0, 1)$. This introduces a systematic mean shift in the problem matrices, testing robustness to distributional mismatch. Figure 8 shows convergence trajectories and Table 8 reports summary statistics.

Table 9: Random least-squares (in-distribution): mean best $f(\boldsymbol{x})$, $\|\nabla f(\boldsymbol{x})\|$, and runtime (seconds) across 100 test instances per dimension. All learned optimizers are trained at n=400 and tested at $n \in \{200, \ldots, 1000\}$. Best learned method in **bold**; † denotes best overall.

	n=200			n=400			n=600			n=800			n=1000		
Method	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time	f	$\|\nabla f\|$	time
L-BFGS	4.63e-4	9.18e-3	0.71	1.06e-4†	9.38e-3†	1.13	2.84e-4	9.65e-3	1.82	1.31e-4	9.58e-3†	2.76	1.70e-4	1.08e-2	21.45
Adam	7.23e-4	1.32e-1	4.89	2.45e-3	5.08e-1	5.06	5.05e-3	1.26e0	5.10	9.04e-3	1.96e0	5.02	9.74e-3	3.05e0	5.07
Nesterov	4.88e-4	1.07e-2	1.11	3.72e-4	1.20e-2	1.34	4.03e-4	1.06e-2	1.65	5.61e-4	1.19e-2	1.92	3.20e-4	1.05e-2	2.60
cLSTM	2.06e-2	5.47e-2	0.09	3.11e-2	7.47e-2	0.09	2.62e4	7.44e3	0.09	6.76e4	1.35e4	0.09	1.19e5	1.96e4	0.09
RNNProp	3.21e0	1.04e1	0.17	4.38e0	2.77e1	0.17	1.79e2	4.96e2	0.18	8.47e2	1.42e3	0.17	1.91e3	2.36e3	0.17
L2O-DM	6.27e-1	1.14e1	0.28	1.21e0	1.85e1	0.28	1.37e0	2.70e1	0.28	3.26e0	3.90e1	0.28	5.28e0	4.99e1	0.28
MINN	**3.42e-4**†	**8.61e-3**†	**0.50**	**1.49e-4**	**9.54e-3**	**0.73**	**2.63e-4**†	**9.50e-3**†	**1.20**	**1.19e-4**†	**9.74e-3**	**1.80**	**1.32e-4**†	**9.82e-3**†	**1.75**

Figure 8 shows that distribution shift amplifies the separation between purely learned optimizers and hybrid classical–learned optimization. When the test distribution is shifted from $\mathcal{N}(0, 1)$ to $\mathcal{N}(0.15, 1)$, cLSTM, RNNProp, and L2O-DM lose stability almost immediately. Their trajectories either stall at poor objective values or increase sharply, indicating that the learned update rules have overfit to the training distribution and do not provide reliable OOD descent behavior.

The classical methods are substantially more robust under the same shift. L-BFGS, Adam, and Nesterov continue to make descent progress across all tested dimensions, confirming that hand-designed optimization mechanisms remain less sensitive to distributional mismatch. However, their efficiency varies: first-order methods can be slow to reach high accuracy, while L-BFGS often achieves strong final accuracy but requires more wall-clock time at larger dimensions.

MINN retains robust convergence under both scale shift and distribution shift. Although trained only at $n_{\text{train}} = 200$ on $\mathcal{N}(0, 1)$ instances, it generalizes to $n_{\text{test}} \in \{400, 600, 800, 1000\}$ under the shifted $\mathcal{N}(0.15, 1)$ distribution. The trajectories show that MINN preserves the early-time acceleration of learned optimizers while avoiding their catastrophic OOD failures. Except at n=400, where Nesterov is slightly faster, MINN reaches low objective values faster than the classical baselines and remains stable throughout the optimization horizon. These results suggest that combining a dimension-agnostic learned update with a learnable L-BFGS refinement under classical safeguards (projection, Armijo search, and the accept/reject filter) yields an optimizer that is both efficient and robust beyond the training distribution.

Empirical stationarity check. We use the stationarity criterion $\|\nabla f(\boldsymbol{x})\| \leq 10^{-2}$ as the target optimality tolerance for all methods. As shown in Tables 7 and 8, MINN's mean best gradient norm meets this criterion at every tested dimension in both the in-distribution and OOD settings. In contrast, the purely learned baselines do not reach this accuracy level, reflecting their tendency to stall or diverge before attaining high-accuracy stationarity. Classical methods are more robust, but their mean best gradient norms often remain above this tolerance: Nesterov's at every tested dimension in both settings, Adam's everywhere except OOD n=200, and L-BFGS's at n=1000 in-distribution and at $n \geq 600$ OOD. These results indicate that MINN preserves the fast initial progress of learned optimizers while maintaining the convergence reliability expected from classical optimization methods, empirically supporting our theoretical convergence analysis.

J.2.2 Training with $n = 400$

We investigate in-distribution generalization by testing below and above the training dimension $n = 400$, using test dimensions $n = 200, 400, 600, 800, 1000$.

Figure 9 shows convergence trajectories and Table 9 reports summary statistics. We also observe that MINN generalizes well not only to larger dimensions but also to lower-dimensional test instances. Although MINN is trained at $n = 400$, it remains stable and efficient when evaluated at $n = 200$. This suggests that the coordinate-wise neural parameterization does not overfit to a fixed problem size, while the classical safeguards preserve reliable descent when the test dimension differs from the training dimension in either direction.

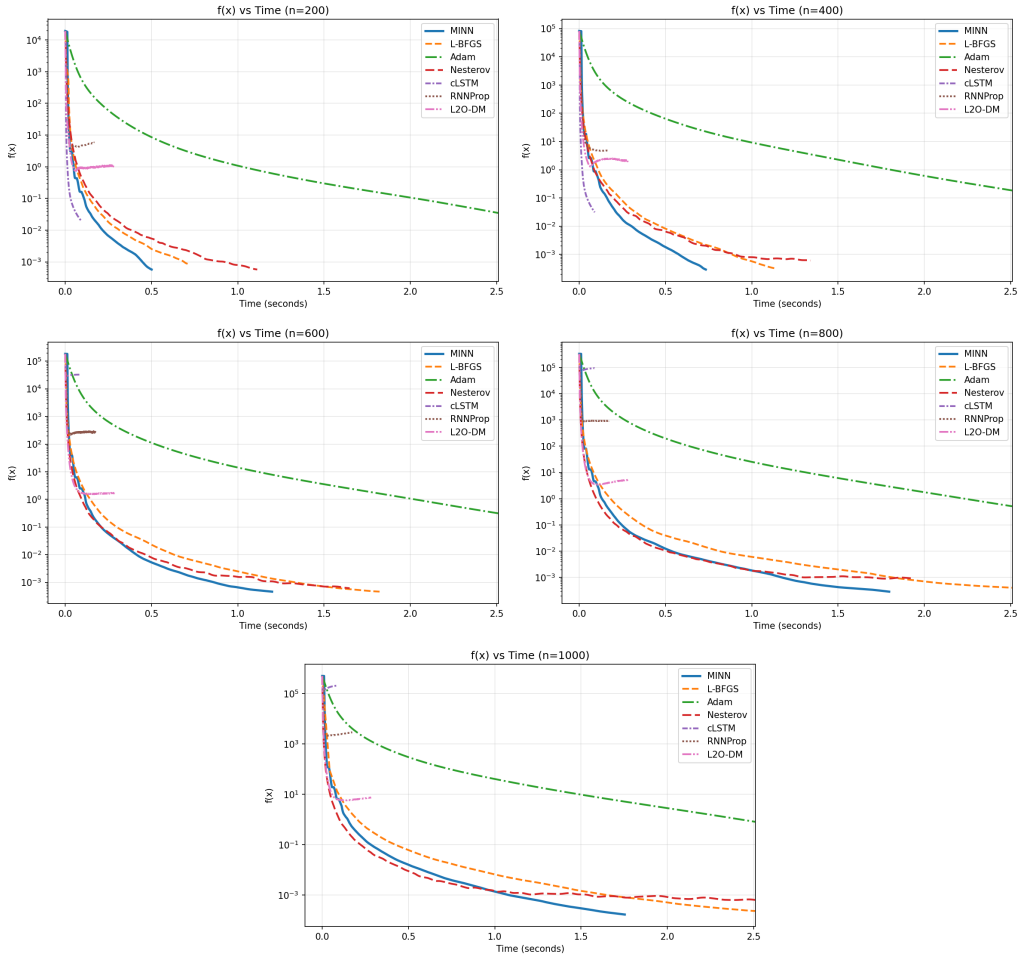

Figure 9: Random least-squares (in-distribution, trained at n=400): convergence of $f(\boldsymbol{x}_t)$ vs. wall-clock time.

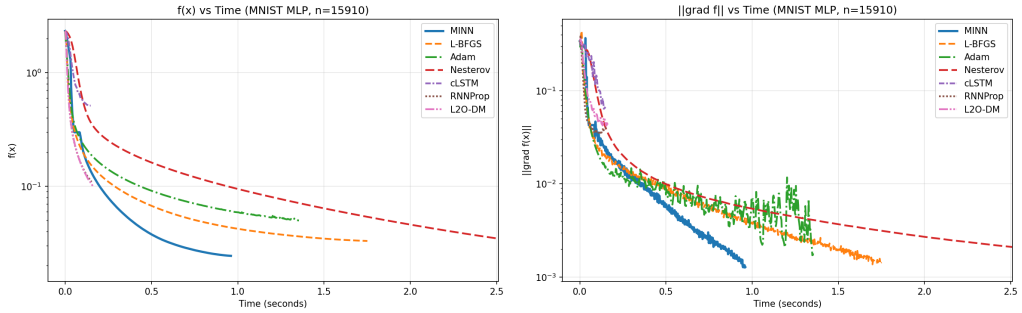

Figure 10: MNIST MLP training (in-distribution): training loss $f(\boldsymbol{x}_t)$ (left) and $\|\nabla f(\boldsymbol{x}_t)\|$ (right).

J.3 Experiment 2: Neural Network Training (MNIST MLP)

In-Distribution Setting. On this high-dimensional (n=15,910) in-distribution task, all optimizers minimize the cross-entropy loss of a $784 \to 20$ (sigmoid) $\to 10$ MLP on MNIST. The task distribution consists of different random initializations; we evaluate on 50 held-out initializations. Figure 10 shows convergence trajectories and Table 10 reports summary statistics.

Table 10: MNIST MLP training (in-distribution): summary statistics over 50 test initializations. Best learned method in **bold**; † denotes best overall.

Method	Mean best f	Std best f	Mean best $\|\nabla f\|$	Mean time (s)	Std time (s)
L-BFGS	3.25e-2	3.35e-3	9.47e-4	1.75	0.19
Adam	4.97e-2	4.36e-3	9.98e-4	1.35	0.04
Nesterov	1.85e-2†	2.01e-3	1.21e-3	4.32	0.04
cLSTM	5.12e-1	1.56e-2	5.67e-2	0.15	0.00
RNNProp	1.10e-1	6.70e-3	4.07e-2	0.15	0.00
L2O-DM	9.93e-2	1.17e-2	4.26e-2	0.16	0.00
MINN	**2.40e-2**	**3.13e-3**	**9.47e-4**†	**0.96**	**0.09**

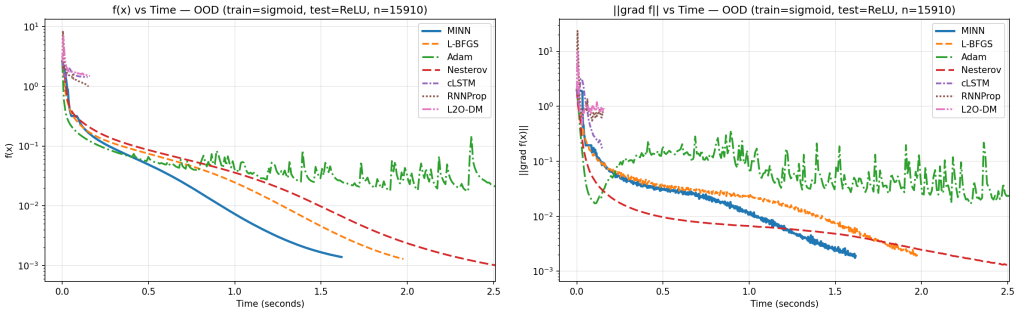

Figure 11: MNIST MLP training (OOD): train on sigmoid, test on ReLU. Training loss $f(\boldsymbol{x}_t)$ (left) and $\|\nabla f(\boldsymbol{x}_t)\|$ (right).

Out-of-Distribution Setting: Sigmoid → ReLU Transfer. To test OOD generalization, all optimizers trained on the sigmoid MLP are evaluated on the same architecture with ReLU activation—a significant change in loss landscape geometry. Figure 11 shows convergence trajectories and Table 11 reports summary statistics.

The MNIST MLP results reveal the same qualitative behavior as in the synthetic experiments. In the matched sigmoid setting, cLSTM, RNNProp, and L2O-DM reduce the loss rapidly at the beginning, confirming the benefit of amortized optimization. However, their curves quickly plateau at substantially worse loss and gradient norm than the classical baselines. Classical methods are slower initially but continue improving, with Nesterov eventually reaching the lowest loss and L-BFGS tying MINN for the lowest gradient norm (Table 10).

MINN combines these two regimes. It inherits the early-time acceleration of the learned optimizers, but its L-BFGS correction prevents the plateau behavior and allows continued progress toward a high-quality stationary point. Under the sigmoid-to-ReLU transfer test, the contrast becomes sharper: purely learned optimizers fail to transfer and remain at high loss, whereas classical methods and MINN remain stable. MINN achieves this robustness while retaining faster convergence than the classical baselines. These results indicate that MINN's learned component provides useful task-level acceleration, while the classical component supplies the numerical structure needed for OOD robustness and final accuracy.

J.4 Summary of Findings

Across the three benchmarks, in distribution and (for least-squares and MNIST) under distribution shift, our experiments consistently demonstrate:

(i) **Speed:** MINN has a lower mean runtime than L-BFGS in every reported setting (by 17–67× on Rastrigin) and than Nesterov in all but three settings (OOD least squares at $n \in \{400, 800, 1000\}$, where Nesterov stops earlier but with mean best gradient norm above 10^{-2}), while achieving comparable or better solution quality.

Table 11: MNIST MLP training (OOD: sigmoid→ReLU): summary statistics over 50 test initializations. Best learned method in **bold**; † denotes best overall.

Method	Mean best f	Std best f	Mean best $\|\nabla f\|$	Mean time (s)	Std time (s)
L-BFGS	8.24e-4	1.14e-4	9.52e-4	1.98	0.31
Adam	5.34e-3	6.68e-3	7.30e-3	4.70	1.11
Nesterov	8.04e-4†	8.68e-5	9.99e-4	2.69	0.14
cLSTM	1.20e0	1.23e-1	1.64e0	0.15	0.00
RNNProp	9.14e-1	2.44e-1	3.43e-1	0.15	0.00
L2O-DM	1.26e0	2.80e-1	5.19e-1	0.16	0.00
MINN	**9.58e-4**	**1.70e-4**	**9.44e-4**†	**1.62**	**0.48**

(ii) **Robustness:** Unlike purely learned optimizers that diverge catastrophically under dimension scaling or distributional shift, MINN remains stable in the reported tests. Its smooth-objective convergence guarantee follows from the projection safeguard, Armijo search, and endpoint filter under the assumptions of Section 4.

(iii) **Generalization:** MINN generalizes gracefully across dimensions (n=200 $\rightarrow$ 1000), across distributions ($\mathcal{N}(0,1) \rightarrow \mathcal{N}(0.15,1)$), and across activation functions (sigmoid→ReLU), validating the hybrid design philosophy.

(iv) **Solution quality:** MINN consistently achieves gradient norms of order 10^{-3}–10^{-2}, indicating approximate first-order stationarity for smooth objectives. The guarantees in Section 4 require their stated assumptions; the ReLU results use the automatic-differentiation diagnostic described in Section 5.

(v) **Training cost:** In every training run we timed (Table 5), MINN trains in under two minutes, and the learned baselines take 1.6–$37.0\times$ as long to train.

These results validate our central thesis: the neural component provides amortized speed and adaptivity, while the classical safeguards provide the stated smooth-objective convergence guarantees. OOD performance is an empirical observation, not a consequence of those guarantees.

K Extended Related Work

MINN connects two established lines of work: safeguarding learned updates and learning components of classical optimization algorithms. The useful comparison is therefore where learning enters, what information it uses, and which test controls the resulting iterate. Table 12 summarizes these mechanisms. The cited methods address different problem classes and use different assumptions; the table does not compare the strength of their convergence guarantees.

How the two stages are connected. Stage 1 rescales the LSTM direction and bounds its mixing weights, so the mixed direction has norm at most $(\tau_{\max} + \nu_{\max})\|\boldsymbol{g}_t\|$. Projection onto $\mathcal{H}_t = \{\boldsymbol{d} : \boldsymbol{g}_t^\top \boldsymbol{d} \le -\kappa\|\boldsymbol{g}_t\|^2\}$ adds the smallest multiple of $-\boldsymbol{g}_t$ that enforces sufficient descent, and Armijo backtracking gives $\hat{\boldsymbol{x}}_t$. Stage 2 receives $\hat{\boldsymbol{x}}_t$ and the trial-step pairs, which consist of displacements and differences of true gradients, not of LSTM states or predicted curvature. Each inner step uses the current gradient, and new pairs replace the initial ones first-in-first-out (Sections 3.4 and D.1). The endpoint is kept only if $f(\boldsymbol{z}_t) \le f(\hat{\boldsymbol{x}}_t)$.

What the convergence analysis establishes. The proofs follow the classical argument for Armijo line search with sufficient-descent directions (Nocedal and Wright, 2006, Ch. 3): the direction bounds give a positive lower bound on the step lengths, and summing the decreases gives $\sum_t \|\boldsymbol{g}_t\|^2 < \infty$. The endpoint test preserves this decrease. Section 4 states the assumptions.

Table 12: Mechanisms in related approaches and their relation to MINN. The distinctions concern the algorithmic construction, not a ranking of performance or convergence guarantees.

Approach	**Learned object or guard**	**Relation to MINN**
Moeller et al. (2019)	Projection of neural updates onto a descent half-space with Armijo-type backtracking; a variant parametrizes the output to satisfy both a sufficient-descent condition and $\|\boldsymbol{d}\| \leq \zeta_2 \|\boldsymbol{g}\|$ for a constant $\zeta_2 > 0$, with a linear rate under the Polyak–Łojasiewicz (PL) inequality (constant step).	MINN obtains the norm bound by rescaling the LSTM direction and mixing it with $-\boldsymbol{g}_t$ through bounded weights, and adds an L-BFGS refinement initialized from the trial steps, with endpoint acceptance.
Heaton et al. (2023)	Learned candidates safeguarded by a fixed-point residual test and a classical fallback in convex optimization.	In Stage 1, MINN modifies the learned direction before the line search instead of choosing between a learned and a classical update.
Prémont-Schwarz et al. (2022)	A loss-based guard compares a learned optimizer with a reference optimizer.	MINN also keeps a fallback; it is the Stage-1 point of the same outer iteration, not the output of a separate optimizer.
Fahy et al. (2024)	Greedy learning of iteration-dependent preconditioning and momentum in gradient-based updates.	MINN trains all components jointly with an unrolled loss and enforces sufficient decrease at every outer iteration at test time.
LODO (Liao et al., 2023)	Quasi-Newton preconditioner represented by a neural network and learned online during the test solve, without meta-training.	MINN trains offline and fixes its parameters at test time; it keeps the two-loop recursion, initializes it with measured gradient differences, and learns memory-slot weights and step lengths.
Banert et al. (2020; 2024)	Learned parameters of convergent proximal schemes (2020); networks that select updates from a set on which convergence is proved (2024).	MINN enforces a direction condition for smooth objectives and tests the refinement endpoint.
MINN	Bounded mixing, half-space projection, Armijo search, and learnable L-BFGS refinement with endpoint acceptance.	Trial-step gradient differences give the initial L-BFGS memory; acceptance keeps the Stage-1 decrease.