NEURAL-REFINE: A LEARN-AND-REFINE FRAMEWORK FOR SCALABLE CONSTRAINED OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning-based solvers can amortize the cost of repeatedly solving related constrained optimization problems, but a neural prediction is useful only if it is both feasible and competitive in objective value. Existing predict-then-correct approaches often train the predictor largely independently of the numerical procedure used later to repair or improve its output, so the learned component need not discover starting points that are especially useful to that downstream optimizer. We introduce NEURAL-REFINE, a closed-loop learn-and-refine framework in which the numerical backend used at deployment also generates training feedback, without differentiating through the solver. The proposer produces multiple candidate starting points and, when useful, compact auxiliary information used by the numerical refiner. Periodically during training, the backend refines these candidates; the best feasible refined solution found so far for each instance is retained as a stop-gradient *self-elite* target for later proposer updates. At deployment, the trained proposer, deterministic refiner, and a common feasibility-first selector are executed once. Across nine benchmark tasks formed by QP, QCQP, and SOCP in convex, nonconvex, and nonsmooth settings, NEURAL-REFINE passes the 10^{-4} feasibility criterion on all nine, attains mean 10^{-6} feasible rate 84.3% versus 2.4% for FS-Net, and gives the best feasible objective among learning-based methods on six tasks. On the tasks where a reference CPU IPOPT configuration meets the same criterion, NEURAL-REFINE also has lower online time, a deployment-level rather than hardware-normalized comparison. In a separate matched-GPU scaling study, numerical solvers are faster at small dimensions, and NEURAL-REFINE becomes faster as the dimension grows. With separate training at each target dimension, a shared-weight sparse realization remains operational up to 10^9 variables on a generated suite with very sparse constraint coupling.

1 INTRODUCTION

Many optimization systems do not solve one isolated problem; they solve a stream of closely related problems whose structure stays fixed while demands, measurements, or operating conditions change. This pattern appears in model-predictive control, resource allocation, and power-system optimization (Sánchez-Sánchez and Izzo, 2018; Ichnowski et al., 2021; Pan et al., 2022). Learning-to-optimize methods exploit this repetition by learning from previously solved instances and using the learned model to reduce online work (Amos, 2023; Kotary et al., 2021; Sambharya et al., 2023). The difficulty is that, for constrained problems, a fast prediction is useful only if it leads to a feasible, high-quality solution. A small constraint violation can invalidate the output, while repairing that violation may move the point enough to change its objective value substantially (Donti et al., 2021; Liang et al., 2024; Nguyen and Donti, 2025).

This makes the numerical routine used after prediction part of the learning problem, not merely an implementation detail. Existing approaches enforce constraints through specialized parameterizations, projection or correction layers, or differentiable solver iterations (Donti et al., 2021; Liang and Chen, 2025; Nguyen and Donti, 2025; Sambharya et al., 2023). These designs show that learned prediction and numerical structure can be complementary. Recent self-training and bootstrapping methods further show that iterative improvement can provide useful learning signal (Lüken and Lucia, 2026; Ding et al., 2026; Nguyen et al., 2026). They also sharpen a practical question: when the *deployed* refiner is deterministic, family-specific, non-differentiable, or too expensive to unroll, how should its

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.

downstream success shape the predictor? In that setting, the best raw prediction need not be the best starting state for the solver that will actually run at inference time.

Our answer is to train the proposer for what happens *after* refinement. We introduce NEURAL-REFINE, a closed-loop learn-and-refine framework in which a learned proposer generates several candidate states and optional compact hints, a deterministic numerical backend refines them using known problem structure, and a feasibility-first rule selects the final solution. During training, we periodically run the same refiner with gradients disabled. For each training instance, the best feasible refined result found so far becomes a stop-gradient *self-elite* target for later proposer updates. In this way, the deployed numerical routine influences what the network learns without being placed inside the backward graph. Figure 1 gives the full loop, and Appendix B.2 provides a two-dimensional walkthrough.

The separation is intentional. The learned component captures recurring instance-to-state structure, while the numerical backend retains the family-specific operations needed for feasibility and local improvement. We use this interface across QP, QCQP, and SOCP. At large scale, numerical solvers rely on sparse, matrix-free, and GPU-oriented computation to control memory and factorization costs (Applegate et al., 2025; Pacaud and Shin, 2024; Montoison et al., 2025), so a dense proposer whose output layer grows with the number of variables would become the bottleneck. For the scale study, we therefore replace the fixed-dimensional proposer by a shared-weight bipartite GNN whose parameter count does not grow with the number of variables. We train a separate checkpoint at each target dimension, so the experiment asks whether the interface remains practical as problems grow, not whether one checkpoint transfers across sizes.

We evaluate NEURAL-REFINE on nine controlled tasks spanning QP, QCQP, and SOCP in convex, nonconvex, and nonsmooth settings. The method passes the 10^{-4} feasibility criterion on all nine tasks, reaches a mean feasible rate of 0.843 at the stricter 10^{-6} tolerance versus 0.024 for FSNet, and obtains the best feasibility-gated objective among learning-based methods on six tasks. On the separate sparse scaling suite, the GNN variant keeps QP/QCQP objective gaps at or below 2.62% wherever a feasible numerical anchor is available (e.g., 0.59% at 10^6-variable QP and 0.77% at 10^7-variable QCQP). Over the same anchored range, the matched-GPU comparison changes from a numerical-solver advantage at small scale to a NEURAL-REFINE advantage at larger scale, and the shared-weight design remains operational through 10^9 variables. CPU IPOPT timings are kept as a separate deployment reference rather than a hardware-normalized speed claim.

Our contributions are:

- **A closed-loop learn-and-refine training mechanism.** The deployed numerical backend periodically converts current proposals into refined feasible outcomes, and successful outcomes become stop-gradient self-elite targets for later proposer updates.
- **A common learned-to-numerical interface across problem classes.** Multiple learned starting states and optional compact hints are combined with deterministic, family-specific refinement; a shared-weight GNN instantiation removes the fixed-dimensional output layer in the scale study.
- **An evaluation of feasibility, objective quality, online time, and scale.** Under one feasibility-first protocol, NEURAL-REFINE passes the 10^{-4} criterion on all nine controlled tasks and is evaluated against learning-based and numerical baselines from the controlled benchmark to a generated sparse suite reaching 10^9 variables.

2 RELATED WORK

2.1 LEARNING-BASED CONSTRAINED OPTIMIZATION

Learning-to-optimize methods learn solution maps or solver states for recurring parametric problems (Amos, 2023; Kotary et al., 2021). Representative approaches predict primal or primal–dual quantities directly, or train from the objective and constraints without optimal labels (Park and Van Hentenryck, 2023; Park et al., 2023; Kotary and Fioretto, 2024). NEURAL-REFINE shares the goal of reducing repeated online work, but treats the learned prediction as an input to a downstream numerical procedure rather than as the final solution.

2.2 Feasibility and Solver Feedback

Constraint handling has been incorporated through completion and correction, feasible parameterizations, learned projection, and feasibility-seeking layers (Donti et al., 2021; Li et al., 2023; Liang et al., 2024; Liang and Chen, 2025; Nguyen and Donti, 2025). Recent methods also make iterative improvement part of learning: LISCO jointly trains a primal–dual predictor and a learned iterative KKT solver (Lüken and Lucia, 2026); DiOpt bootstraps a diffusion solver from feasibility- and objective-weighted candidates (Ding et al., 2026); and Cheap Thrills combines inexpensive labels with subsequent self-supervised refinement (Nguyen et al., 2026). These works make clear that refinement feedback is a broader design pattern; more generally, distilling the best outcome of a slower improvement procedure into a fast policy is the idea behind expert iteration and self-imitation learning (Anthony et al., 2017; Oh et al., 2018). Our specific interface uses the *deployed deterministic backend*—which may be non-differentiable and family-specific—to produce per-instance stop-gradient self-elites, without back-propagating through that backend or using external solver labels. Appendix H gives a more detailed comparison.

2.3 Large-Scale Optimization and Scalable Learned Interfaces

Large numerical solvers are shaped by sparse linear algebra, memory movement, and accelerator efficiency. PDLP targets very large LPs with matrix–vector-based first-order updates (Applegate et al., 2025), while GPU nonlinear and conic solvers move sparse kernels and linear algebra onto accelerators (Pacaud and Shin, 2024; Montoison et al., 2025). Learned optimization has only more recently reached larger structured instances: PDHG-Net reports million-node PageRank experiments (Li et al., 2024), and GNN warm-start methods share parameters across variable–constraint graphs (Schmidtobreick et al., 2025). Our scale study asks a narrower systems question: can the learned interface remain size-independent in parameter count while sparse, family-specific refinement carries the numerical work from QP to QCQP and SOCP?

3 Problem Setting

We consider a distribution of constrained optimization problems

$$z^\star(\xi) \in \arg\min_{z\in\mathbb{R}^n} f(z;\xi) \quad \text{s.t.} \quad h(z;\xi)=0, \quad g(z;\xi)\le 0, \quad \ell \le z \le u, \tag{1}$$

where $z \in \mathbb{R}^n$ is the decision vector and ξ contains the instance-specific parameters. We write $h(z;\xi) \in \mathbb{R}^{m_h}$ and $g(z;\xi) \in \mathbb{R}^{m_g}$ for equality and inequality residuals. Within one task, the dimensions and structural coefficients are fixed while ξ varies; Appendix A gives the QP, QCQP, and SOCP instances used in the benchmark. The scaling suite treats each target dimension n as a separate task.

For a vector y, let $[y]_+$ denote the elementwise positive part. We measure maximum constraint violation by

$$v(z;\xi) = \max\{\|h(z;\xi)\|_\infty, \|[g(z;\xi)]_+\|_\infty, \|[\ell - z]_+\|_\infty, \|[z-u]_+\|_\infty\}. \tag{2}$$

A point is ε-feasible when $v(z;\xi)\le\varepsilon$. We use 10^{-4} as the primary tolerance and report 10^{-6} as a stricter diagnostic. Objective values are compared only after the corresponding feasibility criterion is satisfied; the selector in Section 4.4 follows the same feasibility-first convention.

4 The Learn-and-Refine Framework

4.1 Overview

Given an instance, Neural-Refine first predicts a set of candidate solver states. A deterministic numerical routine then refines each candidate using the known problem structure, and a shared selector returns the best feasible point encountered during refinement, or the least-violating point if no feasible point is found. The key training idea is to reuse this same numerical routine: at periodic refresh steps, its best feasible output for each training instance is stored and used as a stop-gradient target for later proposer updates. The numerical routine is therefore not differentiated through, but its outcomes still influence what the proposer learns.

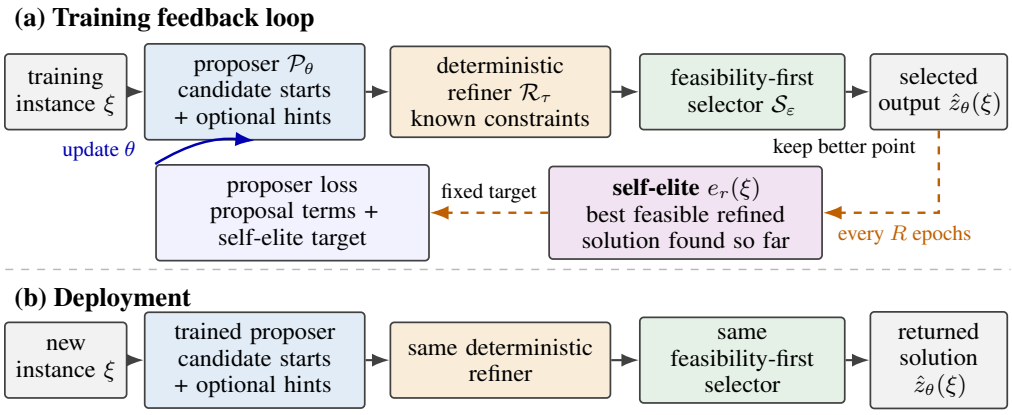


Figure 1: **Training and deployment.** The proposer generates candidate starts, the deterministic refiner improves them, and the selector returns the best feasible point when one exists (otherwise the least-violating point). During training, the same refinement pipeline is run periodically; its best feasible outputs are stored as self-elite targets for subsequent proposer updates.

For an instance ξ, let θ denote the proposer parameters and K the number of candidate starts ($K = 96$ in the fixed-size experiments). The proposer outputs

$$\mathcal{P}_\theta(\xi) = \left(\mathcal{Z}^0_\theta(\xi), q_\theta(\xi)\right), \qquad \mathcal{Z}^0_\theta(\xi) = \{z^0_k(\xi)\}_{k=1}^K, \tag{3}$$

where $z^0_k(\xi) \in \mathbb{R}^n$ is the k-th primal start. The optional $q_\theta(\xi)$ is a family-specific hint: in our fixed-size specialists it is a nonnegative multiplier estimate for KKT/Newton routes, an inequality-activity probability vector for active-set routes, and empty for the SOCP routes.

Let τ index the refinement route, determined by the problem class and setting (Appendix C), and let $\mathcal{R}_\tau$ denote its deterministic refiner. Starting from $z^0_k(\xi)$, the refiner produces a finite trajectory

$$\mathcal{T}_k(\xi) = \mathcal{R}_\tau\left(z^0_k(\xi), q_\theta(\xi); \xi\right) = \{z^0_k(\xi), z^1_k(\xi), \ldots, z^{N_k}_k(\xi)\}, \tag{4}$$

where N_k is the number of refinement steps taken from start k. To compare visited points consistently, define the feasibility-first score

$$\Psi_\varepsilon(z;\xi) = \begin{cases} (0, f(z;\xi)), & v(z;\xi) \le \varepsilon, \\ (1, v(z;\xi)), & v(z;\xi) > \varepsilon, \end{cases} \tag{5}$$

with lexicographic ordering. The selector $\mathcal{S}_\varepsilon$ returns the visited point with the smallest score,

$$\hat{z}_\theta(\xi) = \mathcal{S}_\varepsilon\left(\bigcup_{k=1}^K \mathcal{T}_k(\xi); \xi\right). \tag{6}$$

Thus, any ε-feasible point is preferred to any infeasible point; among feasible points the lower objective is preferred, and among infeasible points the lower violation is preferred. Different problem families may use different numerical refiners; the proposer–refiner–selector interface remains unchanged.

4.2 Refinement-aware training

A loss evaluated only on the initial proposal does not reveal how that proposal behaves after numerical refinement. To connect training to the deployed pipeline, we maintain a *self-elite*: the best feasible refined solution found so far for each training instance.

Let $r = 1, 2, \ldots$ index refresh events and let θ_r denote the proposer parameters at refresh r. We initialize $e_0(\xi) = \varnothing$, use the convention $f(\varnothing;\xi) = +\infty$, and update the self-elite by

$$e_r(\xi) = \begin{cases} \hat{z}_{\theta_r}(\xi), & v(\hat{z}_{\theta_r}(\xi);\xi) \le \varepsilon \text{ and } f(\hat{z}_{\theta_r}(\xi);\xi) < f(e_{r-1}(\xi);\xi), \\ e_{r-1}(\xi), & \text{otherwise.} \end{cases} \tag{7}$$

Hence $e_r(\xi)$ is always either empty or ε-feasible, and once it is feasible its objective cannot increase across refreshes. The stored point is treated as a constant during back-propagation.

Between refreshes, the proposer is trained with a differentiable base loss $\mathcal{L}_{\text{base}}(\xi)$ that measures candidate quality, diversity, and any optional auxiliary prediction. In the reported fixed-size routes, the route-specific structural losses and the self-elite term are attached to a single proposal: after deterministic box/equality preprocessing, the one with the lowest training-time candidate merit M_k (Appendix B.3 gives the exact score and tie-breaking rule). We denote this proposal by $\tilde{z}^0_{\text{dist}}(\xi)$, where the tilde distinguishes it from the raw proposer output in Eq. (3). If a self-elite is available, we use

$$\mathcal{L}_{\text{elite}}(\xi) = \left\| \tilde{z}^0_{\text{dist}}(\xi) - \text{sg}[e_r(\xi)] \right\|_2^2, \tag{8}$$

where $\text{sg}[\cdot]$ denotes stop-gradient; when $e_r(\xi) = \varnothing$, we set $\mathcal{L}_{\text{elite}}(\xi) = 0$. The per-instance proposer objective is

$$\mathcal{L}_{\text{prop}}(\xi) = \mathcal{L}_{\text{base}}(\xi) + \lambda_{\text{elite}} \mathcal{L}_{\text{elite}}(\xi), \qquad \lambda_{\text{elite}} \geq 0. \tag{9}$$

Minibatch training uses the average of $\mathcal{L}_{\text{prop}}(\xi)$ over the instances in the minibatch. This separates the two computations cleanly: gradients update only the proposer, while periodic solver calls provide targets that reflect downstream refinement quality.

Algorithm 1 Refinement-aware proposer training

Require: training set $\mathcal{D}_{\text{tr}}$, validation set $\mathcal{D}_{\text{val}}$, proposer $\mathcal{P}_\theta$, refiner $\mathcal{R}_\tau$, selector $\mathcal{S}_\varepsilon$, refresh period R
  Initialize $e_0(\xi) \leftarrow \varnothing$ for each $\xi \in \mathcal{D}_{\text{tr}}$ and set $r \leftarrow 0$
  for each epoch t **do**
    for each minibatch $\mathcal{B} \subset \mathcal{D}_{\text{tr}}$ **do**
      Generate proposals with $\mathcal{P}_\theta$ and update θ using the minibatch mean of Eq. (9)
    end for
    if $t \bmod R = 0$ **then**
      $r \leftarrow r + 1$; set $\theta_r \leftarrow \theta$
      Run the current proposer–refiner–selector pipeline on $\mathcal{D}_{\text{tr}}$ with gradients disabled
      Update $e_r(\xi)$ for every training instance using Eq. (7)
      Evaluate the same refined pipeline on $\mathcal{D}_{\text{val}}$ and update the checkpoint
    end if
  end for

In the fixed-size experiments, the self-elite memory is refreshed every 150 epochs, i.e., at most seven times in runs of up to 900–1100 epochs, and $\lambda_{\text{elite}} = 0.05$. These refreshes add offline training cost because they execute the numerical backend, but they do not change deployment cost and do not retain a backward graph through the refinement steps. Exact losses, refresh settings, and numerical budgets are given in Appendices B.3 and C.

4.3 Proposer architecture and size-independent parameterization

The proposer is designed to predict states that are useful to the downstream numerical routine. For the standard benchmark, we use task-specific fixed-size networks, matching the setting of prior learned baselines. Each network predicts K primal starts and, for problem families that use them, a compact auxiliary hint. The exact architectures are given in Appendix B.3.

A fixed-dimensional output map is unsuitable for the scale study because its output layer grows with the number of decision variables. We therefore use a bipartite GNN whose variable nodes represent decision variables and whose constraint nodes represent constraints. Nonzero coefficients in the problem data define sparse variable–constraint edges. Message-passing and decoding functions are shared across nodes of the same type. After message passing, the same node-wise decoder is applied to every variable node to produce the corresponding coordinate of a candidate solution.

This parameter sharing makes the neural parameter count independent of the problem dimension (roughly 8×10^4 at every tested n): increasing n adds graph nodes and sparse edges but does not enlarge the learned map, and candidates are decoded and refined in chunks rather than as a dense $K \times n$ tensor. A size-independent parameterization is not zero-shot size generalization; each dimension is trained separately. Graph features and execution details are given in Appendix B.4.

4.4 Family-specific refinement and feasibility-first selection

The deterministic backend enforces known mathematical structure and performs local numerical improvement. QP, QCQP, and SOCP use refiners matched to their constraints and derivatives (Appendix C). These routines need not be differentiable or share internal iterations; each only maps a proposed state to a finite set of refined candidate points. This modularity is not backend-free generality: a new constraint family still requires a compatible numerical refiner.

All families share the selector $\mathcal{S}_\varepsilon$, which applies the score in Eq. (5) to every visited point, so the final comparison rule is identical across routines. The self-elite update in Eq. (7) is stricter: it stores only feasible refined points, never an infeasible fallback. At deployment, θ is frozen, no self-elite memory is maintained, and each instance passes once through the proposer, refiner, and selector.

5 Experiments

We evaluate feasibility, objective quality, online time, and scalability: first on nine standard QP/QCQP/SOCP tasks against learning-based solvers and IPOPT, then on a sparse suite from 10^3 to 10^9 variables against GPU numerical solvers. Complete tables and scale records are in Appendices D and E.

5.1 Experiment settings

Problem classes and settings. The standard suite comprises QP, QCQP, and SOCP in smooth convex (C), smooth nonconvex (NC), and nonsmooth nonconvex (NS) settings. Each task uses 1000 instances with an $800/100/100$ train/validation/test split. Each instance has 100 variables, 50 equalities, and 50 inequalities. Structural matrices are fixed within a task; only the equality right-hand side varies. This fixed-size benchmark aligns baseline comparisons; the separate sparse suite tests scaling. Appendix A gives the formulations. Appendix G adds a separate AC optimal power flow (AC-OPF) study on PGLib-OPF networks (Babaeinejadsarookolaee et al., 2021) that pairs a GNN warm start with GPU MadNLP.

Baselines. We compare with FSNet (Nguyen and Donti, 2025), Penalty, Adaptive Penalty (Fioretto et al., 2020), DC3 (Donti et al., 2021), Projection (Amos and Kolter, 2017), bisection projection (BP) (Liang and Chen, 2025), homeomorphic projection (H-Proj) (Liang et al., 2024), and IPOPT (Wächter and Biegler, 2006) in one fixed, untuned reference configuration (limited-memory Hessian, neutral initialization; Appendix F). FSNet is the strongest learning-based baseline with complete nine-task coverage in our aligned suite. The reported Neural-Refine models use no baseline-derived objective value, solution, trajectory, dual variable, warm start, or solver call during training (Appendix F). For large-scale numerical comparisons, we use MadNLP (Pacaud and Shin, 2024) and MadIPM (Montoison et al., 2025) on GPU.

Metrics. For returned solutions $\hat{z}_i$ on a test set of size N, we report

$$F_\varepsilon = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{v(\hat{z}_i;\xi_i) \le \varepsilon\}\,, \tag{10}$$

with $F_4 := F_{10^{-4}}$ and $F_6 := F_{10^{-6}}$. A standard-suite task passes when $F_\varepsilon \ge 0.999$, i.e., all $N = 100$ test points are feasible. Objectives are compared only after the F_4 gate, and runtime is reported per test instance. Appendix F gives the full conventions and uncertainty intervals.

Large-scale evaluation. At each target dimension, a separate model is trained and evaluated on a $90/10/5$ train/validation/test split. Its structural rows scale as roughly $\lceil\sqrt{n}\rceil$, with $\lceil\log_2 n\rceil$ nonzeros per sparse row. The generator is therefore very sparse: at $n = 10^9$, the roughly 3.2×10^4 structural rows touch at most about 9.5×10^5 coordinates ($<0.1\%$), so large-n instances are weakly coupled and chiefly test whether the learned interface and sparse refiner scale in memory and time, not whether they solve harder coupled problems. We use the five held-out instances to test completion, feasibility, online time, and the objective gap, i.e., the signed relative objective difference to a feasible numerical anchor where one exists (negative means lower than the anchor); Appendix A.1 gives the exact generator.

5.2 COMPARISON WITH LEARNING-BASED CONSTRAINED SOLVERS

Table 1 gives two representative convex tasks with the broad baseline set. On convex QP, NEURAL-REFINE matches the best feasible objective, reaches $F_4 = F_6 = 1.00$, and solves each instance in 0.043 s; FSNet also passes F_4, but only 1% of its outputs pass 10^{-6}. On convex SOCP, NEURAL-REFINE again reaches $F_4 = F_6 = 1.00$, but its objective is 1.1% above the IPOPT optimum and above FSNet's at the 10^{-4} gate; it is best only among learned methods with $F_6 = 1$. The refiner thus secures strict feasibility on both tasks, while objective quality is optimal on QP but not on SOCP.

Table 1: Convex QP and SOCP results. Obj. is gated by F_4; F_4/F_6 are feasible rates at $10^{-4}/10^{-6}$; Time is seconds per instance. Best feasible objectives are underlined; feasibility maxima are bold. Projection is unavailable on SOCP.

(a) Convex QP

Method	Obj. ↓	F_4 ↑	F_6 ↑	Time ↓
NEURAL-REFINE	-36.2742	**1.00**	**1.00**	0.0430
FSNet	-36.2608	**1.00**	0.01	0.0687
Projection	-36.2590	**1.00**	**1.00**	0.8563
BP	-36.0100	**1.00**	0.55	0.0004
H-Proj	-35.4650	**1.00**	**1.00**	0.0203
DC3	–	0.00	0.00	0.0140
Penalty	–	0.00	0.00	0.0003
Adaptive Pen.	–	0.00	0.00	0.0003
IPOPT	-36.2742	**1.00**	**1.00**	13.7921

(b) Convex SOCP

Method	Obj. ↓	F_4 ↑	F_6 ↑	Time ↓
NEURAL-REFINE	-13.9708	**1.00**	**1.00**	0.0067
FSNet	-14.0568	**1.00**	0.00	0.0850
Projection	–	–	–	–
BP	-13.9190	**1.00**	0.96	0.0002
H-Proj	-13.4260	**1.00**	**1.00**	0.0027
DC3	–	0.50	0.50	0.0157
Penalty	–	0.00	0.00	0.0003
Adaptive Pen.	–	0.00	0.00	0.0004
IPOPT	-14.1272	**1.00**	**1.00**	3.1121

Across all nine tasks, the main advantage is consistency rather than a single favorable case. NEURAL-REFINE is the only evaluated method to pass the primary 10^{-4} criterion on all nine tasks (FSNet: 8/9; IPOPT: 5/9 in the fixed reference configuration). Tightening the tolerance separates the methods much more sharply: mean F_6 is 0.843 for NEURAL-REFINE and 0.024 for FSNet, with means of 0.937 and 0.927 on the nonconvex and nonsmooth subsets. The gain is not in speed: the median online time of NEURAL-REFINE (0.128 s) exceeds FSNet's (0.084 s; Appendix D).

Feasibility does not come at the cost of uniformly worse objectives. Among learning-based methods, NEURAL-REFINE gives the best F_4-gated objective on six of nine tasks, including four of the six nonconvex/nonsmooth cases; FSNet is lower on the other three (C-SOCP, NC-QCQP, NS-QCQP). These wins differ in strength: on the three QPs and C-QCQP the margin over FSNet is below 0.1%, and on NS-SOCP FSNet has no F_4-feasible objective; the clearest objective gain is on NC-SOCP (-2.79 versus -1.18).

Table 2: Results on all nine tasks. "–" marks F_4 failure. Objectives are rounded to two decimals; underlining marks the best feasible objective among displayed methods using unrounded values. Bold marks the highest F_6. †Below IPOPT only through residual violation ($F_6 = 0$; see text).

		Convex			Nonconvex			Nonsmooth			
Method	Metric	QP	QCQP	SOCP	QP	QCQP	SOCP	QP	QCQP	SOCP	Mean
NEURAL-REFINE	Obj. ↓	-36.27	-26.93†	-13.97	-19.75	-17.92	-2.79	-18.90	-16.85	-1.40	–
	F_6 ↑	**1.00**	0.00	**1.00**	0.95	**0.86**	**1.00**	**0.90**	**0.90**	**0.98**	**0.84**
FSNet	Obj. ↓	-36.26	-26.91	-14.06	-19.73	-18.09	-1.18	-18.89	-16.89	–	–
	F_6 ↑	0.01	0.00	0.00	0.03	0.03	0.04	0.04	0.07	0.00	0.02
IPOPT	Obj. ↓	-36.27	-26.93	-14.13	-19.75	–	-1.83	–	–	–	–
	F_6 ↑	**1.00**	**1.00**	**1.00**	**1.00**	0.00	**1.00**	0.00	0.00	0.00	0.56

The nonconvex and nonsmooth QCQPs, where IPOPT fails F_4, make the trade-off concrete: FSNet reaches slightly lower feasible objectives, while NEURAL-REFINE raises F_6 from 0.03 to 0.86 and from 0.07 to 0.90. On NC-SOCP, NEURAL-REFINE improves both strict feasibility and the feasible

objective, and on NS-SOCP it is the only displayed method to pass F_4. C-QCQP is the clear exception: every returned point satisfies 10^{-4}, but none satisfies 10^{-6}, and its objective falls marginally below IPOPT's convex optimum (both -26.929) only because of these residual violations, not because it finds a better solution. We therefore treat 10^{-4} as the supported operating point of this route.

The refinement diagnostic in Appendix E makes the division of labor concrete: with the proposer fixed, mean F_4 over four representative tasks (C-QP, C-QCQP, NC-QP, NS-QP) rises from 0.315 before refinement to 0.998 after it.

5.3 Online solution time and scaling

Three patterns emerge as the problem size grows. First, QP/QCQP objective gaps stay small wherever a feasible numerical anchor exists: at most 2.62% in the convex setting (e.g., 0.59% for C-QP at 10^6 and 0.77% for C-QCQP at 10^7) and at most 0.66% for all recorded NC/NS QP and QCQP points. SOCP is less uniform (11.58% at 10^4, 3.52% at 10^5). Gaps tend to shrink with n, but because coupling weakens as n grows (Section 5.1), this is not evidence that harder problems become easier for the method.

Second, runtime crosses over while those anchors are still available (Figure 2(b)–(d), convex setting). Matched-GPU numerical solvers are faster at small n; for QP and QCQP the crossover lies between 10^5 and 10^6 variables, where NEURAL-REFINE reaches $2.46\times$ and $5.05\times$ speedups at 0.59% and 0.49% gaps. For SOCP it comes earlier, at 10^4 ($1.12\times$, but at the largest gap, 11.58%), and grows to $50.1\times$ at 10^5 (3.52% gap). The QP anchors at 10^5 and 10^6 come from MadNLP, a general nonlinear solver, rather than the QP-specific MadIPM used at 10^3. All ratios compare wall time at unequal objective accuracy, not time to a common target, and the timeout/OOM markers are observation boundaries, not inferred runtimes. Figure 2(a) separately reports $31.2\times$–$1252.2\times$ ratios against CPU IPOPT on the standard suite; we use these only as deployment references, not as hardware-normalized algorithmic comparisons.

Third, beyond the last anchor, all nine tasks still complete all five held-out 10^9-variable instances with $F_4 = 1$, taking 196–255 s for QP/QCQP and 432–1085 s for SOCP. No objective gap is reported there, so Table 3 separates anchored objective quality at verifiable scales from feasibility and runtime at the frontier.

Table 3: Scaling results for all nine tasks: at $n = 10^9$, F_4 (as a count over the five test instances) and online time; at smaller n, objective gaps (%) to reliable feasible numerical anchors. "–" means no anchor; negative nonconvex gaps indicate lower local objectives, without certifying global optimality. 10^4 is omitted for space; it contains the largest gap (C-SOCP, 11.58%; Table 11).

		10^9 frontier		Anchored objective gap (%)			
Regime	Family	F_4	Time (s)	10^3	10^5	10^6	10^7
C	QP	5/5	225.1	1.34	0.43	0.59	–
	QCQP	5/5	255.1	2.62	1.17	0.49	0.77
	SOCP	5/5	431.7	1.95	3.52	–	–
NC	QP	5/5	196.2	0.62	0.33	0.29	0.17
	QCQP	5/5	199.2	0.43	0.23	0.44	0.17
	SOCP	5/5	1083.0	-0.56	-9.27	–	–
NS	QP	5/5	198.4	0.66	0.23	0.20	–
	QCQP	5/5	202.7	0.00	0.20	0.28	–
	SOCP	5/5	1085.0	-0.54	–	–	–

6 Discussion and Limitations

Attribution. The fixed-proposer diagnostic shows that refinement is essential, but no experiment yet isolates self-elite training: a matched retraining with $\lambda_{\text{elite}} = 0$ and an equal-work random or heuristic multistart through the same refiner are still missing (Appendix F). Without them, the gains

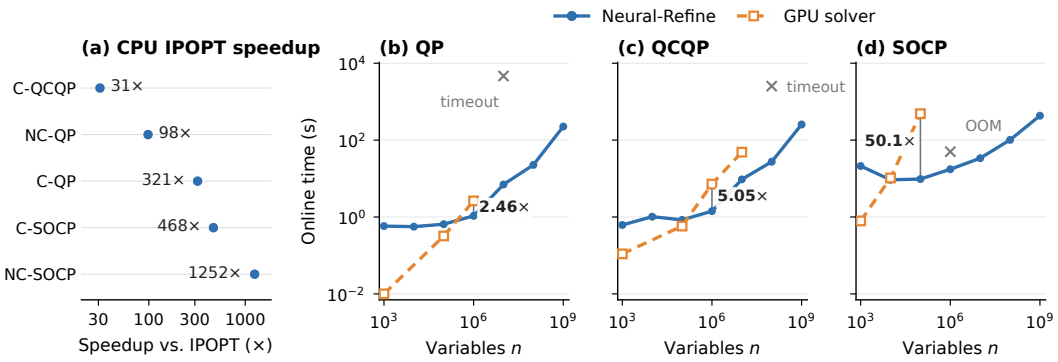


Figure 2: **Online solution time and scaling.** (a) Deployment-time ratio against CPU IPOPT on the five standard-suite tasks ($n = 100$) where IPOPT passes F_4. (b)–(d) NEURAL-REFINE versus matched-GPU MadIPM/MadNLP anchors on the convex scaling suite (shared y-axis). Only successful solver runs are plotted; the observed QP timeout and SOCP OOM boundaries are marked, and no missing solver point is extrapolated.

cannot be attributed to the closed-loop mechanism rather than to learned multistart refinement in general.

Baselines and timing. CPU IPOPT is run in one untuned configuration and is not hardware-matched to NEURAL-REFINE, the GPU anchors are general second-order interior-point solvers (MadNLP even for the larger QPs) rather than structure-exploiting first-order GPU solvers, and speed ratios are taken at unequal objective accuracy; all ratios are therefore deployment references. Online times also exclude the per-scale training cost, which must be amortized over many solves.

Scope. The standard suite is small ($n = 100$) and varies only the equality right-hand side; each task uses one selected checkpoint, so seed variance is not estimated; the large-n suite is weakly coupled with five test instances per scale; each dimension is trained separately, so we make no zero-shot transfer claim; and a new constraint family requires a compatible refiner. Transfer across sizes and structures, denser large-scale generators, and less family-specific refinement are natural next steps.

7 CONCLUSION

NEURAL-REFINE trains neural proposals from refined feasible outcomes without differentiating through the deployed backend. Across nine controlled QP/QCQP/SOCP tasks, it passes the primary 10^{-4} criterion on all nine and gives the best feasibility-gated learned objective on six. In the scale study, QP/QCQP objective gaps remain small while reliable numerical anchors exist, matched-GPU runtime crosses over as n grows, and the shared sparse GNN remains operational through 10^9 variables on a sparse generated suite with roughly 8×10^4 learned parameters. These results support pairing a learned proposer with numerical refinement so that the learned model is not itself responsible for final feasibility; quantifying how much the self-elite loop adds beyond multistart refinement is the most important next step.

Reproducibility Statement

The benchmark formulations, proposer and refinement details, evaluation rules, and additional aggregate results are documented in Appendices A–F. 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.

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

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32, 2019.

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023.

Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34:20243–20257, 2021.

David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. PDLP: A practical first-order method for large-scale linear programming. *arXiv preprint arXiv:2501.07018*, 2025.

Sogol Babaeinejadsarookolaee, Adam Birchfield, Richard D. Christie, Carleton Coffrin, Christopher DeMarco, Ruisheng Diao, Michael Ferris, Stephane Fliscounakis, Scott Greene, Renke Huang, Cedric Josz, Roman Korab, Bernard Lesieutre, Jean Maeght, Terrence W. K. Mak, Daniel K. Molzahn, Thomas J. Overbye, Patrick Panciatici, Byungkwon Park, Jonathan Snodgrass, Ahmad Tbaileh, Pascal Van Hentenryck, and Ray Zimmerman. The power grid library for benchmarking AC optimal power flow algorithms, 2021. URL https://arxiv.org/abs/1908.02788.

Shutong Ding, Yimiao Zhou, Ke Hu, Xi Yao, Junchi Yan, Xiaoying Tang, and Ye Shi. Diffusion-based learning framework for constrained nonconvex optimization with weighted bootstrapped refinement. In *International Conference on Machine Learning*, 2026.

Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

Ferdinando Fioretto, Pascal Van Hentenryck, Terrence WK Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. Lagrangian duality for constrained deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 118–135. Springer, 2020.

Xi Gao, Jinxin Xiong, Akang Wang, Qihong Duan, Jiang Xue, and Qingjiang Shi. IPM-LSTM: A learning-based interior point method for solving nonlinear programs. *Advances in Neural Information Processing Systems*, 37:122891–122916, 2024.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. *Advances in Neural Information Processing Systems*, 34:21043–21055, 2021.

James Kotary and Ferdinando Fioretto. Learning constrained optimization with deep augmented Lagrangian methods. *arXiv preprint arXiv:2403.03454*, 2024.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, 2021.

Bingheng Li, Linxin Yang, Yupeng Chen, Senmiao Wang, Qian Chen, Haitao Mao, Yao Ma, Akang Wang, Tian Ding, Jiliang Tang, et al. PDHG-unrolled learning-to-optimize method for large-scale linear programming. *arXiv preprint arXiv:2406.01908*, 2024.

Meiyi Li, Soheil Kolouri, and Javad Mohammadi. Learning to solve optimization problems with hard linear constraints. *IEEE Access*, 11:59995–60004, 2023.

Enming Liang and Minghua Chen. Efficient bisection projection to ensure neural-network solution feasibility for optimization over general set. In *ICML*, 2025.

Enming Liang, Minghua Chen, and Steven H Low. Homeomorphic projection to ensure neural-network solution feasibility for constrained optimization. *Journal of Machine Learning Research*, 25(329):1–55, 2024.

Steven H. Low. Convex relaxation of optimal power flow—part I: Formulations and equivalence. *IEEE Transactions on Control of Network Systems*, 1(1):15–27, 2014. doi: 10.1109/TCNS.2014.2309732. URL https://arxiv.org/abs/1405.0766.

Anqi Lu and Junchi Yan. Learning initial basis selection for linear programming via duality-inspired tripartite graph representation and comprehensive supervision. In *ICML*, 2025.

Lukas Lüken and Sergio Lucia. Self-supervised learning of iterative solvers for constrained optimization. *Results in Control and Optimization*, page 100751, 2026.

Alexis Montoison, François Pacaud, Sungho Shin, and Mihai Anitescu. GPU implementation of second-order linear and nonlinear programming solvers. *arXiv preprint arXiv:2508.16094*, 2025.

Hoang Nguyen and Priya Donti. FSNet: Feasibility-seeking neural network for constrained optimization with guarantees. *Advances in Neural Information Processing Systems*, 38:39670–39708, 2025.

Khai Nguyen, Petros Ellinas, Anvita Bhagavathula, and Priya L Donti. Cheap Thrills: Effective amortized optimization using inexpensive labels. *arXiv preprint arXiv:2603.05495*, 2026.

NVIDIA. *NVIDIA cuOpt User Guide*, 2026. URL https://docs.nvidia.com/cuopt/user-guide/latest/index.html. Version 26.08.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning*, pages 3878–3887. PMLR, 2018.

François Pacaud and Sungho Shin. GPU-accelerated dynamic nonlinear optimization with ExaModels and MadNLP. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 5963–5968. IEEE, 2024.

Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H Low. DeepOPF: A feasibility-optimized deep neural network approach for AC optimal power flow problems. *IEEE Systems Journal*, 17(1): 673–683, 2022.

Seonho Park and Pascal Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4052–4060, 2023.

Seonho Park, Wenbo Chen, Terrence WK Mak, and Pascal Van Hentenryck. Compact optimization learning for AC optimal power flow. *IEEE Transactions on Power Systems*, 39(2):4350–4359, 2023.

Alban Puech, Matteo Mazzonelli, Tamara R. Govindasamy, Mangaliso Mngomezulu, Héctor Maeso-García, Thomas Tolhurst, Javad Bayazi, Ali Moeini, Naomi Simumba, Celia Cintas, David Nelischer, Romeo Kienzler, Jonas Weiss, Anna Varbella, Florian Dörfler, Gabriela Hug, Martin Mevissen, Juan Bernabé-Moreno, François Mirallès, Hendrik F. Hamann, Etienne Vos, and Thomas Brunschwiler. GENCO – a unified neural solver embedded in a development framework for steady-state grid analysis, 2026. URL https://arxiv.org/abs/2608.09921.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Learning for dynamics and control conference*, pages 220–234. PMLR, 2023.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024.

Carlos Sánchez-Sánchez and Dario Izzo. Real-time optimal control via deep neural networks: study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5):1122–1135, 2018.

Ella J Schmidtobreick, Daniel Arnström, Paul Häusner, and Jens Sjölund. Warm-starting active-set solvers using graph neural networks. *arXiv preprint arXiv:2511.13174*, 2025.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

A Benchmark Formulations

All controlled-benchmark tasks use $z \in \mathbb{R}^{100}$, 50 affine equalities $Az = b$, 50 structural inequalities, and lower/upper bounds. Here $A \in \mathbb{R}^{50\times 100}$ is the equality matrix and $b \in \mathbb{R}^{50}$ is the varying equality right-hand side; the remaining tensors are fixed within each task. Elementwise trigonometric functions are written as $\sin z$ and $\cos z$, and $\odot$ denotes elementwise multiplication.

QP. Let $Q \in \mathbb{R}^{100\times 100}$ and $p \in \mathbb{R}^{100}$ denote the fixed quadratic and linear objective coefficients. The convex objective is

$$f_{\mathrm{c}}(z) = \tfrac{1}{2} z^\top Q z + p^\top z, \tag{11}$$

while the nonconvex and nonsmooth objectives are

$$f_{\mathrm{nc}}(z) = \tfrac{1}{2} z^\top Q z + p^\top \sin z, \qquad f_{\mathrm{ns}}(z) = f_{\mathrm{nc}}(z) + 0.1\|z\|_2. \tag{12}$$

Let $G \in \mathbb{R}^{50\times 100}$ and $c \in \mathbb{R}^{50}$ denote the fixed inequality coefficients. The convex residual vector is $Gz - c$, while the nonconvex variants use $G \sin z - c \odot \cos b$.

QCQP. For inequality $j \in \{1, \ldots, 50\}$, let $H_j \in \mathbb{R}^{100\times 100}$, $a_j \in \mathbb{R}^{100}$, and $c_j \in \mathbb{R}$ denote fixed coefficients. The convex residual is

$$g_j(z) = \tfrac{1}{2} z^\top H_j z + a_j^\top z - c_j, \tag{13}$$

whereas nonconvex and nonsmooth variants replace the linear term $a_j^\top z$ by $a_j^\top \cos z$. Objectives follow the QP definitions above.

SOCP. For cone constraint j, let r_j denote its cone dimension and let $G_j \in \mathbb{R}^{r_j\times 100}$, $s_j \in \mathbb{R}^{r_j}$, $c_j \in \mathbb{R}^{100}$, and $d_j \in \mathbb{R}$ be fixed coefficients. The residual is

$$g_j(z) = \|G_j a(z) + s_j\|_2 - (c_j^\top z + d_j), \tag{14}$$

with $a(z) = z$ in the convex task and $a(z) = \cos z$ in the nonconvex and nonsmooth tasks. The nonsmooth objective again adds $0.1\|z\|_2$.

A.1 Sparse scale generator

The scaling suite is generated separately from the fixed-size benchmark. For a target dimension n, the generator uses approximately $\lceil \sqrt{n} \rceil$ structural constraint rows, and each sparse row contains $\lceil \log_2 n \rceil$ nonzero variable couplings. Quadratic terms are diagonal where permitted by the family generator, and SOCP constraints are represented as independent sparse cone blocks. These choices keep storage tied to the sparse problem description rather than to dense $n \times n$ matrices.

For every class, setting, and target dimension, the generator produces 105 aligned instances with a deterministic 90/10/5 train/validation/test split. A separate proposer checkpoint is trained and selected at each n. The five held-out instances are intended to verify completion, feasibility, and observed runtime at extreme scale; they are not used to estimate a high-precision population success rate.

B Training and Proposer Details

B.1 Closed-loop training schedule

The proposer is updated by ordinary back-propagation on proposal-level losses. Every 150 epochs in the fixed-size benchmark, the complete deployed proposer–refiner–selector pipeline is run with gradient tracking disabled. For each training instance, the stored self-elite is updated exactly as in Eq. (7). Later proposer updates treat that stored point as a constant target through Eq. (8). We use $\tilde{z}^0_{\mathrm{dist}}(\xi)$ for the proposal to which the route-specific self-distillation rule is applied; its exact merit-based selection rule is defined in Appendix B.3. Thus, numerical refinement contributes training targets without entering the backward graph. Algorithm 1 gives the main-loop pseudocode. At deployment, the self-elite memory and its update are removed.

B.2 A CONCRETE WALKTHROUGH OF THE TRAINING LOOP

This example is intentionally small and is meant only to make the feedback loop concrete; the numbers below are illustrative rather than benchmark measurements. Suppose a training instance ξ has a two-dimensional decision vector. After the proposer output is passed through the deterministic box/equality preprocessing used by the training loss, assume the candidate selected for self-distillation is

$$\tilde{z}^0_{\mathrm{dist}}(\xi) = (1.30, -1.10).$$

During an ordinary training step, the proposer is updated using its differentiable proposal losses and, if a self-elite is already available, the self-elite loss. The numerical backend is not part of this backward pass.

At a periodic refresh, the current proposal is instead passed through the same numerical routine used at deployment, with gradient tracking disabled. For example, the refiner may produce the sequence

$$(1.30, -1.10) \to (1.70, -0.90) \to (2.00, -0.75) \to (2.08, -0.71).$$

Assume that the last point is ε-feasible and improves on the previously stored feasible solution for this instance. The memory is then updated to

$$e_r(\xi) = (2.08, -0.71).$$

No gradient is propagated through this refinement trajectory.

On later ordinary training steps, $e_r(\xi)$ is treated as a fixed target. The self-elite term contains

$$\left\| \tilde{z}^0_{\mathrm{dist}}(\xi) - \mathrm{sg}[e_r(\xi)] \right\|_2^2,$$

so gradients flow through the current preprocessed proposal $\tilde{z}^0_{\mathrm{dist}}(\xi) = \tilde{z}^0_{\mathrm{dist}}(\xi; \theta)$ but not through the stored refined point. The proposer is therefore trained not only from the quality of its raw output, but also from a feasible solution that the deployed numerical routine was able to reach from that output.

The distinction between training and deployment is then simple. During training, the feedback cycle is

$$\text{propose} \to \text{refine} \to \text{update self-elite} \to \text{update proposer},$$

repeated periodically. At deployment, the memory update is removed and a new instance follows only

$$\text{propose} \to \text{refine} \to \text{feasibility-first select}.$$

This example corresponds directly to the two paths shown in Figure 1.

B.3 FIXED-SIZE PROPOSER AND TRAINING SIGNALS

For the controlled fixed-size benchmark, the varying affine-equality right-hand side is $b = b(\xi)$ and the proposer input is

$$\phi(b) = [b; \mathrm{mean}(b); \mathrm{std}(b); \max(b); \min(b)]\,. \tag{15}$$

The reported specialists use a 384-dimensional hidden representation with four gated residual blocks and $K = 96$ learned mode embeddings sharing a primal head. KKT-oriented routes additionally predict a nonnegative multiplier vector $\mu(\xi) \in \mathbb{R}^{m_g}_+$, active-set routes predict activity probabilities $p^{\mathrm{act}}(\xi) \in [0,1]^{m_g}$, and SOCP uses the primal candidates alone. In the notation of Eq. (3), $q_\theta(\xi)$ is $\mu(\xi)$, $p^{\mathrm{act}}(\xi)$, or empty for these three cases, respectively. All fixed-size specialists use double precision.

For the losses below, $\tilde{z}^0_k(\xi)$ denotes the k-th proposal after the deterministic box/equality preprocessing used by the fixed training route; $z^0_k(\xi)$ without a tilde remains the raw proposer output from Eq. (3). Because these candidates have already undergone equality correction, the differentiable candidate merit penalizes the remaining inequality violation:

$$M_k(\xi) = f(\tilde{z}^0_k(\xi); \xi) + \rho_k \left(\|[g(\tilde{z}^0_k(\xi); \xi)]_+\|_\infty + \|[g(\tilde{z}^0_k(\xi); \xi)]_+\|_2^2 \right), \tag{16}$$

where $\rho_k > 0$ is the feasibility weight assigned to mode k. The soft best-of-K weights and candidate-quality loss are

$$M_{\min}(\xi) = \min_{1 \le j \le K} M_j(\xi), \tag{17}$$

$$\omega_k(\xi) = \frac{\exp[-(M_k(\xi) - M_{\min}(\xi))]}{\sum_{j=1}^{K} \exp[-(M_j(\xi) - M_{\min}(\xi))]}, \tag{18}$$

$$\mathcal{L}_{\text{cand}}(\xi) = \sum_{k=1}^{K} \omega_k(\xi) M_k(\xi). \tag{19}$$

The candidate used by the route-specific structural losses and the self-elite term is selected by this same merit:

$$k_{\text{dist}}(\xi) = \arg\min_{1 \le k \le K} M_k(\xi), \qquad \tilde{z}^0_{\text{dist}}(\xi) = \tilde{z}^0_{k_{\text{dist}}(\xi)}(\xi), \tag{20}$$

with ties broken by the smallest candidate index. Because ω_k is monotone in $-M_k$, this is equivalently the proposal with the largest soft best-of-K weight. To discourage collapse of the portfolio, let $\bar{z}^0(\xi) = K^{-1} \sum_{k=1}^{K} \tilde{z}^0_k(\xi)$ and define

$$\mathcal{L}_{\text{div}}(\xi) = \frac{1}{K} \sum_{k=1}^{K} \|\tilde{z}^0_k(\xi) - \bar{z}^0(\xi)\|_2^2. \tag{21}$$

The diversity quantity is used as a reward, i.e., it enters the corresponding loss with a negative coefficient.

For KKT-oriented specialists, let $J_g(z;\xi) = \nabla_z g(z;\xi) \in \mathbb{R}^{m_g \times n}$ denote the Jacobian of g and define the regularized equality-tangent operator

$$P_A = I_n - A^\top (AA^\top + \delta I_{m_h})^{-1} A. \tag{22}$$

where I_n and I_{m_h} are identity matrices of the indicated sizes and δ is the same regularization used by the equality correction in Appendix C. For the route-selected proposal $\tilde{z}^0_{\text{dist}}(\xi)$, the auxiliary losses are

$$\mathcal{L}_{\text{stat}}(\xi) = \left\| P_A \left(\nabla_z f(\tilde{z}^0_{\text{dist}}(\xi);\xi) + J_g(\tilde{z}^0_{\text{dist}}(\xi);\xi)^\top \mu(\xi) \right) \right\|_2^2, \tag{23}$$

$$\mathcal{L}_{\text{comp}}(\xi) = \sum_{j=1}^{m_g} \mu_j(\xi) [-g_j(\tilde{z}^0_{\text{dist}}(\xi);\xi)]_+ + \sum_{j=1}^{m_g} [g_j(\tilde{z}^0_{\text{dist}}(\xi);\xi)]_+. \tag{24}$$

For active-set specialists, with near-active margin $\eta_{\text{act}} > 0$, the geometric target is

$$y^{\text{act}}_j(\xi) = \mathbf{1}\left\{ g_j(\tilde{z}^0_{\text{dist}}(\xi);\xi) > -\eta_{\text{act}} \right\}, \qquad \mathcal{L}_{\text{act}}(\xi) = \text{BCE}(p^{\text{act}}(\xi), y^{\text{act}}(\xi)), \tag{25}$$

where BCE denotes the standard binary cross-entropy aggregated over the m_g inequality coordinates. These auxiliary targets are computed from the current problem geometry; they are not external optimal primal or dual labels.

The fixed-route objectives used in the reported experiments are

$$\mathcal{L}_{\text{KKT}}(\xi) = \mathcal{L}_{\text{cand}}(\xi) - 0.02\mathcal{L}_{\text{div}}(\xi) + 0.08\mathcal{L}_{\text{stat}}(\xi) + 0.08\mathcal{L}_{\text{comp}}(\xi) + 0.05\mathcal{L}_{\text{elite}}(\xi), \tag{26}$$

$$\mathcal{L}_{\text{AS}}(\xi) = \mathcal{L}_{\text{cand}}(\xi) + 0.1\mathcal{L}_{\text{act}}(\xi) + 0.05\mathcal{L}_{\text{elite}}(\xi), \tag{27}$$

Cone specialists use the corresponding candidate-quality objective and the same self-elite mechanism when enabled. No baseline-derived objective value, solution, trajectory, dual variable, warm start, or solver call is used by these training objectives. The main text groups the route-specific terms into $\mathcal{L}_{\text{base}}$ so that the closed-loop mechanism can be stated without tying the framework to one specialist loss.

B.4 Scale-up GNN proposer

The scaling experiments replace the fixed-width output map by a bipartite GNN with shared parameters across variable nodes and constraint nodes. Family-specific sparse node and edge features are constructed from the corresponding QP, QCQP, or SOCP tensors. Message-passing and update functions are shared across all nodes of the same type, and each variable node is decoded through the same local scalar output map rather than through a dense $\mathrm{Linear}(h, n)$ layer. Consequently, the trainable parameter count is independent of the number of variables; the reported model contains roughly 8×10^4 parameters for every tested n.

This construction is size-invariant at the architectural level: increasing the problem dimension changes the graph size and sparse feature tensors, but not the dimensions of the learned layers. It therefore avoids retracing the network architecture for each variable count. The experiments nevertheless train a separate model at each target dimension, so the scaling study asks whether the same architecture can be trained and deployed at increasing sizes rather than whether one checkpoint transfers zero-shot across sizes.

For each target dimension, the GNN is self-trained on 90 instances and selected on 10 validation instances; the final five aligned rows are reserved for testing. The candidate violation term is normalized by the number of constraints so that the scale of the feasibility penalty does not grow merely because the graph is larger. The largest runs use minibatch accumulation, chunked candidate decoding, activation checkpointing, mixed precision inside the neural backbone when required for memory, and tensor parallelism along the variable dimension. Candidate chunks are decoded and consumed by the sparse numerical path without materializing a dense $K \times n$ portfolio in device memory. Objective accumulation, explicit constraint correction, and numerical refinement remain in high precision.

B.5 Optimization and checkpoint selection

Training uses Adam with learning rate 2×10^{-3} and gradient clipping at norm 10. Fixed-size specialist training runs for up to 900–1100 epochs depending on the problem class. At each 150-epoch refresh, the full refined pipeline is run under `no_grad` on the training instances to update self-elites and on validation instances for checkpoint selection. This cost is incurred only during training; gradients are computed only for the proposer updates between refreshes. The large-scale proposer uses the shared GNN described above, keeping the learned parameter count fixed while a separate model is trained at each target dimension.

C Numerical Refinement Details

C.1 Benchmark specialization

In the controlled benchmark, affine equalities have the form $Az = b(\xi)$. The reported implementation uses the regularized equality-correction map

$$\Pi_{\mathrm{eq}}(z;\xi) = z - A^\top (AA^\top + \delta I_{m_h})^{-1}(Az - b(\xi))\,, \tag{28}$$

with $\delta = 10^{-10}$. When $\delta = 0$ and A has full row rank, this reduces to the Euclidean projection onto $\{z : Az = b(\xi)\}$; with the reported positive regularization it is a numerically stabilized correction rather than an exact projection. Box clipping is followed by equality re-correction. The remaining local refinement uses the known structure of each task rather than asking the neural model to reproduce that algebra.

For KKT/Newton routes, the predicted nonnegative multipliers enter the partial-Lagrangian stationarity term used to form the local KKT system and help prioritize near-active inequality rows. For active-set QCQP, the predicted activity probabilities are combined with residual-based activity to initialize the working set before restoration/SQP updates. These hints are advisory: feasibility checks and acceptance remain deterministic, and SOCP routes use no learned auxiliary state.

The QP/Newton routes use a fixed 40-step budget per retained start, with damping and a small set of local perturbations. Active-set QCQP uses up to 80 descent steps, with an additional restoration prepass for the nonsmooth case. SOCP uses fixed-grid line search and five continuation stages for

Table 4: Numerical refinement used by the reported specialists. The learned component supplies starting points and, where useful, compact auxiliary information; the numerical stage enforces the problem structure.

Task group	Optional auxiliary information	Deterministic refinement
QP (all settings)	nonnegative multipliers μ	safeguarded multistart Newton/KKT
Convex QCQP	nonnegative multipliers μ	quadratic-constraint Newton/KKT
NC/NS QCQP	activity probabilities p^{act}	restoration + active-set SQP
SOCP (all settings)	none	cone-safe refinement / continuation

the nonconvex/nonsmooth settings. These are fixed numerical hyperparameters rather than learned quantities.

C.2 Feasibility-first selection

All routes use the common selector in Eq. (5), evaluated with the same objective and maximum-violation definition in Eq. (2). The self-elite update in Eq. (7) stores only feasible points, whereas the deployment selector may return the least-violating point when no feasible iterate has been visited.

Proposition 1 (Monotonicity of the stored feasible incumbent)**.** *Under the feasibility-first rule, once an ε-feasible incumbent has been stored, no infeasible point can replace it. Among ε-feasible candidates, its objective is nonincreasing as additional iterates or starts are examined.*

Proof. Every feasible point has first lexicographic component 0, whereas every infeasible point has first component 1. Between two feasible points, the second component is the objective, so replacement occurs only when the new feasible objective is lower.

D Additional Benchmark Results

This appendix expands the standard-suite comparison in Section 5.2. The central NEURAL-REFINE/FSNet/IPOPT rows use the seed-2025 reproduction reported in the main text. The broader baseline comparison additionally includes Projection, BP, H-Proj, DC3, Penalty, and Adaptive Penalty from the aligned full-baseline evaluation. Unsupported cells are shown as "–" and are never imputed.

D.1 Aggregate feasibility and online time

Table 5: Aggregate results for methods evaluated on all nine controlled tasks. "Feasible tasks" counts tasks with feasible rate at least 0.999. Median time is per-instance online solution time in seconds.

Method	At 10^{-4}		At 10^{-6}		Median time
	Feasible tasks	Mean rate	Feasible tasks	Mean rate	(s)
Penalty	0/9	0.000	0/9	0.000	0.00030
Adaptive Penalty	0/9	0.000	0/9	0.000	0.00045
DC3	0/9	0.230	0/9	0.230	0.01570
FSNet	8/9	0.889	0/9	0.024	0.08435
NEURAL-REFINE	**9/9**	**1.000**	3/9	**0.843**	0.12836
IPOPT	5/9	0.556	5/9	0.556	3.99866

D.2 COMPLETE COMPARISONS BY PROBLEM CLASS

The following three tables report objective, F_4, and F_6 for QP, QCQP, and SOCP, respectively. To keep the presentation consistent with the feasibility-gated protocol, the objective column is shown as "–" whenever a method fails the task-level F_4 gate; raw failed-row outputs are retained only in the machine-readable evaluation artifacts.

Table 6: QP results. Each task block reports Obj. / F_4 / F_6.

	C-QP			NC-QP			NS-QP		
Method	Obj.	F_4	F_6	Obj.	F_4	F_6	Obj.	F_4	F_6
NEURAL-REFINE	-36.274	1.00	1.00	-19.746	1.00	0.95	-18.904	1.00	0.90
FSNet	-36.261	1.00	0.01	-19.733	1.00	0.03	-18.889	1.00	0.04
IPOPT	-36.274	1.00	1.00	-19.753	1.00	1.00	–	0.00	0.00
Projection	-36.259	1.00	1.00	–	0.01	0.01	–	0.01	0.01
BP	-36.010	1.00	0.55	–	–	–	–	–	–
H-Proj	-35.465	1.00	1.00	–	–	–	–	–	–
DC3	–	0.00	0.00	–	0.07	0.07	–	0.47	0.47
Penalty	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00
Adaptive	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00

Table 7: QCQP results. Each task block reports Obj. / F_4 / F_6.

	C-QCQP			NC-QCQP			NS-QCQP		
Method	Obj.	F_4	F_6	Obj.	F_4	F_6	Obj.	F_4	F_6
NEURAL-REFINE	-26.929	1.00	0.00	-17.918	1.00	0.86	-16.846	1.00	0.90
FSNet	-26.909	1.00	0.00	-18.085	1.00	0.03	-16.892	1.00	0.07
IPOPT	-26.929	1.00	1.00	–	0.00	0.00	–	0.00	0.00
Projection	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00
BP	-26.390	1.00	0.79	–	–	–	–	–	–
H-Proj	-25.629	1.00	1.00	–	–	–	–	–	–
DC3	–	0.02	0.02	–	0.06	0.06	–	0.07	0.07
Penalty	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00
Adaptive	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00

Table 8: SOCP results. Each task block reports Obj. / F_4 / F_6.

	C-SOCP			NC-SOCP			NS-SOCP		
Method	Obj.	F_4	F_6	Obj.	F_4	F_6	Obj.	F_4	F_6
NEURAL-REFINE	-13.971	1.00	1.00	-2.788	1.00	1.00	-1.402	1.00	0.98
FSNet	-14.057	1.00	0.00	-1.179	1.00	0.04	–	0.00	0.00
IPOPT	-14.127	1.00	1.00	-1.831	1.00	1.00	–	0.00	0.00
Projection	–	–	–	–	–	–	–	–	–
BP	-13.919	1.00	0.96	–	–	–	–	–	–
H-Proj	-13.426	1.00	1.00	–	–	–	–	–	–
DC3	–	0.50	0.50	–	0.76	0.76	–	0.12	0.12
Penalty	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00
Adaptive	–	0.00	0.00	–	0.00	0.00	–	0.00	0.00

D.3 CPU IPOPT ONLINE-TIME COMPARISON

Table 9: CPU IPOPT comparison on tasks where IPOPT itself passes F_4. Ratio is IPOPT time divided by NEURAL-REFINE online solution time and is computed from the unrounded timings.

Task	NEURAL-REFINE time (s)	IPOPT time (s)	Ratio
C-QP	0.0430	13.7921	320.7×
C-QCQP	0.1284	3.9987	31.2×
C-SOCP	0.0067	3.1121	467.6×
NC-QP	0.1315	12.9047	98.1×
NC-SOCP	0.0116	14.5812	1252.2×

E FRAMEWORK DIAGNOSTIC AND SCALING DETAILS

E.1 EFFECT OF NUMERICAL REFINEMENT ON FEASIBILITY

We hold the proposer checkpoint fixed and compare its raw output with the output obtained after the deployed deterministic refinement stage. This isolates the effect of refinement on feasibility without changing the learned proposer. It does not, by itself, isolate the learned portfolio from an equal-work random multistart baseline or isolate self-elite training from a matched no-self-elite model.

Table 10: Feasible rate F_4 before and after the deployed numerical refinement stage on four representative tasks. The proposer checkpoint is fixed.

Output	C-QP	C-QCQP	NC-QP	NS-QP	Mean
Proposal only	0.26	0.53	0.25	0.22	0.315
After numerical refinement	0.99	1.00	1.00	1.00	0.998

E.2 OBJECTIVE QUALITY THROUGHOUT THE SCALE SWEEP

The anchored gap is the evaluator-reported signed relative objective difference to a feasible numerical anchor at the same scale. We report it only when such an anchor is available. A negative value in a nonconvex problem means that NEURAL-REFINE reached a lower local objective than the anchor; it is not a certificate of global optimality.

Table 11: Convex scaling records. Every listed run completes 5/5 held-out instances and is feasible at 10^{-4}. Gap is reported only where a reliable feasible numerical anchor exists.

QP

n	Gap (%)	Time (s)
10^3	1.34	0.58
10^4	0.51	0.56
10^5	0.43	0.65
10^6	0.59	1.07
10^7	–	7.02
10^8	–	22.63
10^9	–	225.1

QCQP

n	Gap (%)	Time (s)
10^3	2.62	0.62
10^4	1.17	1.02
10^5	1.17	0.84
10^6	0.49	1.42
10^7	0.77	9.50
10^8	–	27.40
10^9	–	255.1

SOCP

n	Gap (%)	Time (s)
10^3	1.95	21.21
10^4	11.58	9.26
10^5	3.52	9.71
10^6	–	17.45
10^7	–	33.82
10^8	–	101.3
10^9	–	431.7

Table 12: Selected nonconvex and nonsmooth scaling records. Each entry is anchored objective gap (%) / NEURAL-REFINE online time (s). A dash means no reliable feasible objective anchor at that scale. All listed runs are F_4-feasible.

Family	10^3	10^5	10^6	10^7	10^9
Nonconvex					
QP	0.62/5.40	0.33/6.40	0.29/9.40	0.17/63.5	–/196.2
QCQP	0.43/6.80	0.23/7.30	0.44/10.3	0.17/66.9	–/199.2
SOCP	-0.56/89.7	-9.27/89.0	–/115.5	–/635.6	–/1083.0
Nonsmooth					
QP	0.66/5.70	0.23/7.00	0.20/9.40	–/64.7	–/198.4
QCQP	0.00/6.20	0.20/7.30	0.28/10.7	–/66.9	–/202.7
SOCP	-0.54/90.5	–/92.0	–/118.8	–/640.6	–/1085.0

E.3 MATCHED-GPU NUMERICAL ANCHORS AND STOPPING RECORDS

Table 13: Successful GPU numerical anchors used in Figure 2. Gap is the anchored objective gap; ratio is numerical-solver time divided by NEURAL-REFINE time, so values above one favor NEURAL-REFINE. Only F_4-feasible anchors are included.

Class	n	Gap (%)	NEURAL-REFINE (s)	Solver (s)	Solver	Ratio
QP	10^3	1.34	0.58	0.01	MadIPM	0.02×
QP	10^5	0.43	0.65	0.32	MadNLP	0.49×
QP	10^6	0.59	1.07	2.63	MadNLP	2.46×
QCQP	10^3	2.62	0.62	0.11	MadNLP	0.18×
QCQP	10^5	1.17	0.84	0.58	MadNLP	0.69×
QCQP	10^6	0.49	1.42	7.17	MadNLP	5.05×
QCQP	10^7	0.77	9.50	48.54	MadNLP	5.11×
SOCP	10^3	1.95	21.21	0.79	MadNLP	0.04×
SOCP	10^4	11.58	9.26	10.36	MadNLP	1.12×
SOCP	10^5	3.52	9.71	486.7	MadNLP	50.12×

Table 14: Observed numerical-solver stopping boundaries. Missing or unsuccessful solver runs are not converted into speedups. For QCQP, no larger-scale failure mode is inferred beyond the last recorded successful anchor.

Class	Attempted n	NEURAL-REFINE time (s)	Numerical-solver observation
QP	10^7	7.02	timeout ($>$ 4660 s recorded wall time)
SOCP	10^6	17.45	OOM after approximately 50 s

F EVALUATION AND REPRODUCIBILITY NOTES

Splits, metrics, and test-instance uncertainty. The evaluator uses the seed-2025 canonical indices and aligns methods to the same test instances. On the standard suite, $N = 100$ and the task-level threshold $F_\varepsilon \geq 0.999$ is therefore equivalent to all 100 returned points satisfying the tolerance. Objectives are arithmetic means over aligned instances and are compared only after the method passes F_4; failed-row raw outputs remain in the machine-readable artifacts but are not displayed as objectives in the paper. The main tables report selected checkpoints and therefore do not estimate training-seed variance.

For reference, Table 15 gives 95% Wilson intervals for the per-task F_6 proportions. These intervals quantify finite-test-set uncertainty conditional on the selected checkpoints; they do not replace multi-seed retraining.

Table 15: Per-task strict-feasibility rates with 95% Wilson intervals over the $N = 100$ test instances.

Task	NEURAL-REFINE F_6	95% CI	FSNet F_6	95% CI
C-QP	1.00	[0.963, 1.000]	0.01	[0.002, 0.054]
C-QCQP	0.00	[0.000, 0.037]	0.00	[0.000, 0.037]
C-SOCP	1.00	[0.963, 1.000]	0.00	[0.000, 0.037]
NC-QP	0.95	[0.888, 0.978]	0.03	[0.010, 0.085]
NC-QCQP	0.86	[0.779, 0.915]	0.03	[0.010, 0.085]
NC-SOCP	1.00	[0.963, 1.000]	0.04	[0.016, 0.098]
NS-QP	0.90	[0.826, 0.945]	0.04	[0.016, 0.098]
NS-QCQP	0.90	[0.826, 0.945]	0.07	[0.034, 0.138]
NS-SOCP	0.98	[0.930, 0.995]	0.00	[0.000, 0.037]

CPU IPOPT reference. The standard-size IPOPT row is an evaluator-faithful reference, not a tuned or learned warm-start baseline. It uses cyipopt/Ipopt on the same objective, constraints, bounds, and neutral task initialization as the benchmark; the reported configuration uses a limited-memory Hessian, MUMPS, $\texttt{tol} = 10^{-8}$, and $\texttt{max_iter} = 3000$. Nonsmooth tasks are passed to IPOPT without smoothing, and returned points are scored by the common evaluator even when the solver does not report success. Hence the five-of-nine F_4 coverage characterizes this fixed reference configuration rather than IPOPT universally.

Per-scale training and sample size. The scaling study trains a separate model at each target dimension. The shared-weight architecture keeps the learned parameter count roughly constant with n; the experiment therefore evaluates separate training and deployment at each scale, not zero-shot size transfer. Each target scale has five held-out instances. We report frontier success as a count (5/5), not as a precise population success rate; the generator and sparsity rules are specified in Appendix A.1.

Baseline-independent training. The reported models are trained without baseline-derived objective values, solution vectors, trajectories, dual variables, warm starts, or solver calls. In particular, FSNet is used only as an evaluation baseline. All self-elites are produced by NEURAL-REFINE's own proposer–refiner–selector pipeline on the training instances.

Causal scope and missing controls. The proposal-versus-refinement diagnostic in Appendix E holds the proposer fixed and shows that adding the deployed numerical stage changes feasibility substantially. It does not isolate the self-elite training signal, portfolio size K, refresh frequency, diversity regularization, or λ_{elite}. A matched no-self-elite retraining control and an equal-total-work random/heuristic multistart comparison are therefore the most important missing ablations. We do not substitute unmatched exploratory runs for those controls.

Reproduction package. The anonymous package is intended to include task definitions, canonical splits, selected checkpoints, the evaluator, reported result rows, and scripts that regenerate the main tables and figures. Unsupported baseline/task cells are left unreported rather than imputed.

G AC-OPF DEPLOYMENT ACCELERATION OVER CPU IPOPT

This appendix reports a separate AC optimal power flow (AC-OPF) study (Babaeinejadsarookolaee et al., 2021), outside the controlled nine-task benchmark, to test the same learn-and-refine design pattern on a realistic nonlinear power-system problem. AC-OPF is an important and challenging real-world optimization class: it minimizes generation cost while meeting electricity demand, nonlinear AC power-balance equations, and voltage, generator, and transmission-line limits. Its nonconvex electrical couplings make obtaining high-quality feasible solutions difficult, especially on large

networks. Standard rectangular-coordinate formulations admit nonconvex QCQP representations, with auxiliary power-flow variables where needed (Low, 2014).

Recent learn-to-optimize results illustrate the remaining scale–accuracy challenge. GENCO's reported OPF benchmarks cover networks with 14–2000 buses (Puech et al., 2026); this count refers to buses, rather than decision variables. On the 2000-bus OPFData case, GENCO Base reports mean active and reactive power-balance residuals of 5.87×10^{-2} MW and 2.23×10^{-2} MVar, respectively (Puech et al., 2026, Section 5.2, Table 5). These results motivate evaluating larger networks with explicit feasibility checks. The reported residuals are averages in physical units, so comparison with the maximum-violation tolerance used below requires matched normalization and aggregation.

AC-OPF instantiation and protocol. We use a rectangular current-based AC-OPF formulation over PGLib-OPF cases. The learned component is a size-invariant bipartite GNN that predicts a primal starting point, and the numerical backend is MadNLP with GPU sparse linear algebra (Pacaud and Shin, 2024). The GNN prediction is used only as an initialization; feasibility and the final objective are determined by the numerical refiner. The GPU pipeline uses one fixed configuration, FP64 arithmetic on an H100 GPU, and tolerance 10^{-6}. Each process performs an untimed warm-up solve so reported timings exclude one-time Julia compilation. The CPU reference uses IPOPT (Wächter and Biegler, 2006) on the same current-based formulation. We compute deployment speedup as

$$S_{\mathrm{IPOPT}} = \frac{T_{\mathrm{IPOPT,CPU}}}{T_{\mathrm{GNN+MadNLP,GPU}}}. \tag{29}$$

Because the two pipelines use different hardware and numerical backends, S_{IPOPT} should be read as an end-to-end deployment-time ratio rather than a hardware-normalized comparison of solver algorithms.

Deployment-time acceleration. Among the 53 cases with a reliable IPOPT objective anchor, the GNN+MadNLP pipeline is feasible on all five held-out instances per case at the 10^{-6} tolerance. The median deployment-time ratio is $99.4\times$, with a maximum of $1583.6\times$. Objective quality remains close to the IPOPT anchors: the reported gap is approximately zero on most cases, with the largest anchored gap equal to 0.32%. The speed advantage grows with problem size: the median ratio is about $4.5\times$ below 500 buses, $97\times$ from 500–3000 buses, $172\times$ from 3000–10,000 buses, and roughly $980\times$ at 10,000 buses and above.

Table 16: Representative large AC-OPF cases. "Gap" is the relative objective gap to the feasible CPU IPOPT anchor reported by the common AC-OPF evaluator. Feasibility is evaluated at 10^{-6} over five held-out instances. Speedup is the deployment-time ratio in Eq. (29).

Case	Buses	Feas.	Gap (%)	Ours (s)	CPU IPOPT (s)	Speedup
9241_pegase	9,241	5/5	0.00	1.00	918.16	919.6×
13659_pegase	13,659	5/5	0.00	1.21	1488.54	1232.5×
19402_goc	19,402	5/5	0.00	1.58	1550.30	979.5×
20758_epigrids	20,758	5/5	0.00	1.21	1913.76	1583.6×
30000_goc	30,000	5/5	0.01	3.03	3009.08	993.0×

Table 16 shows the large-grid regime directly. On `20758_epigrids`, for example, online wall time drops from 1913.76 s with CPU IPOPT to 1.21 s while retaining 5/5 feasibility and a near-zero anchored objective gap. Across the listed large cases, minute-scale CPU solves become roughly one-to-three-second online solves.

How much of the gain comes from learning? The CPU-IPOPT ratios measure the complete deployment stack and therefore combine the GPU backend with the learned initialization. To isolate learning, we compare GNN warm starts against flat starts using the *same* MadNLP GPU backend, formulation, tolerance, and timing protocol. On cold-hard cases with useful GNN initializations, the warm start reduces wall time by 7.5–22.2×: for example, `1951_rte` (22.2×), `13659_pegase` (14.7×), `1888_rte` (10.7×), and `2868_rte` (7.5×). It can also enable convergence: at 10^{-6}

tolerance, flat-start MadNLP solves only 1/5 held-out instances of `1888_rte`, whereas the GNN-initialized run solves 5/5 in about 0.54 s. On well-conditioned cases where the cold GPU solver already converges quickly, the warm start is typically neutral and can be slightly slower. Thus, the roughly 10^2–$10^3\times$ CPU-IPOPT ratios should be read as deployment gains of the complete GNN+GPU-refiner stack, while the same-backend comparison shows where the learned proposer adds further value.

H Extended Related Work

Learning solution maps and enforcing feasibility. Learning-based optimization includes direct solution prediction, primal–dual prediction, and self-supervised objectives derived from the optimization model (Kotary et al., 2021; Amos, 2023; Park and Van Hentenryck, 2023; Park et al., 2023; Kotary and Fioretto, 2024). Explicit constraint mechanisms range from DC3's completion/correction pipeline (Donti et al., 2021) and feasible parameterizations (Li et al., 2023) to learned homeomorphic or bisection projections (Liang et al., 2024; Liang and Chen, 2025) and FSNet's differentiable feasibility-seeking procedure (Nguyen and Donti, 2025). These methods differ in their geometric assumptions and in how tightly correction is coupled to training.

Iterative feedback and self-training. Several recent approaches are especially relevant to the feedback-loop perspective. LISCO predicts an initial primal–dual state and then applies a *learned* iterative solver trained from KKT residuals without pre-solved optimizer labels (Lüken and Lucia, 2026). DiOpt uses weighted bootstrapping to improve diffusion-generated candidates toward feasible, high-quality regions (Ding et al., 2026). Cheap Thrills starts from inexpensive imperfect labels and subsequently refines the surrogate through self-supervised optimization (Nguyen et al., 2026). These works overlap with the broad idea that later optimization outcomes can improve learning. Neural-Refine differs in the object being reused: the actual deployment backend is deterministic and need not be differentiable or learned, and its best feasible refined point is cached as a per-instance stop-gradient target. The reported training also uses no external solver or baseline labels.

Learning components of numerical solvers. Other work learns states or updates inside established algorithms. OptNet and differentiable optimization layers expose solver computations to backpropagation (Amos and Kolter, 2017; Agrawal et al., 2019); warm-start methods learn initial states for fixed-point algorithms (Sambharya et al., 2023; 2024); and IPM-LSTM learns updates in an interior-point pipeline (Gao et al., 2024). PDHG-Net unrolls PDHG and hands the predicted primal–dual state to PDLP, including million-node PageRank experiments (Li et al., 2024). Graph models have also been used to predict LP bases or QP active sets (Lu and Yan, 2025; Schmidtobreick et al., 2025). Together these results support learning an algorithmically useful state rather than requiring a network to output the final optimizer solution.

Large-scale numerical optimization. Traditional solver engineering reaches a substantially larger scale regime. PDLP replaces repeated factorizations with matrix–vector operations for very large LPs (Applegate et al., 2021; 2025), while GPU nonlinear/conic solvers move derivative evaluation and sparse linear algebra to accelerators (Pacaud and Shin, 2024; Montoison et al., 2025). Industrial systems similarly emphasize memory footprint, sparse data movement, and models with millions of variables or constraints (NVIDIA, 2026). Our scale study does not propose a new numerical solver; it tests whether the *learned interface* avoids becoming the bottleneck. Shared variable/constraint-node updates keep the neural parameter count roughly constant, while the family-specific numerical backend carries the sparse optimization work. Separate checkpoints are trained at each dimension, so the claim is scalable parameterization and execution rather than zero-shot generalization.