LEARNING TO OPTIMIZE ACROSS PROBLEM SCALES: SIZE-AGNOSTIC SOLVER SELECTION FOR ACCELERATED CONSTRAINED OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard numerical optimization typically fixes a single solver for the entire iterations, even though the local geometry of an optimization trajectory can shift across well-conditioned and ill-conditioned regimes with different computational tradeoffs. We formulate solver selection as a sequential decision process and learn a size-agnostic selection function that dynamically chooses between different solvers along a single trajectory. The selection function operates on a fixed-dimensional, problem-size-independent state representation, allowing one shared model to generalize across problem scales and structures without scale-specific retraining, in contrast to existing learning-to-optimize (L2O) methods that are usually scale-specific. We also propose an asynchronous selection mechanism to reduce the additional overhead of solver selection. Theoretically, we derive a critical condition number that governs local solver preference and show that heterogeneous condition-number regimes induce a positive solver-selection margin. We also characterize how selection errors under distribution shift translate into runtime regret and give a sufficient condition for improvement over static single-solver baselines. Experiments on nonconvex ACOPF and convex QP benchmarks show that our selector generalizes zero-shot across problem scales and unseen structures, while improving convergence reach and reducing wall-clock runtime by 15% on QP and 6% on ACOPF over existing L2O baselines.

1 INTRODUCTION

Constrained numerical optimization is central to scientific computing and machine learning, yet standard solvers are typically fixed for the entire iterations (Boyd et al., 2011). This ignores a key property of optimization trajectories: even for a single problem instance, the local KKT geometry can shift across well-conditioned and ill-conditioned regimes, leading to that solver changing is computationally more favorable. In well-conditioned regimes, per-iteration computationaly efficient methods such as ADMM (alternating direction method of multipliers) can make effective progress (Stellato et al., 2020), and the per-iteration overhead of heavy second-order methods such as PD-IPM (primal-dual interior point method) is excessively high (Wright, 1997). In ill-conditioned regimes with anisotropic curvature, however, cheap updates may converge slowly. First-order gradient methods, for example, can zig-zag (Nocedal & Wright, 2006), whereas heavy solvers can exploit curvature to reduce the number of iterations. Consequently, the relative advantage of cheap and heavy solvers is state-dependent rather than instance-level constant: static single-solver execution is necessarily suboptimal on some stages of the trajectory. This motivates treating solver execution as a sequential, state-dependent selection problem (Adriaensen et al., 2022), as illustrated in Fig. 1.

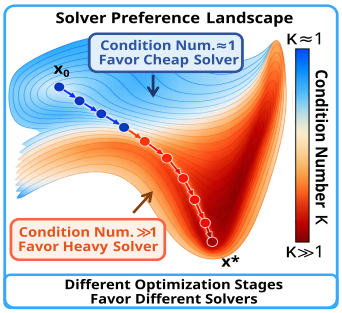


Figure 1: Solver preference landscape across optimization stages.

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.

To enable automatic solver selection, we learn a selection function that observes the current optimization state and decides whether to apply a cheap solver $\mathcal{F}$ (ADMM) or a heavy solver $\mathcal{S}$ (PD-IPM). The objective is to minimize the total wall-clock time to reach a prescribed convergence criterion, not merely the number of iterations. This state-dependent formulation allows the solver to switch along a single optimization trajectory as the local geometry changes.

Learning such a selection function is challenging because local progress alone is an unreliable feature: a cheap update may reduce the residual quickly yet still lead to slow tail convergence, while a heavy update may appear to make significant progress per iteration but still requires substantial runtime due to its excessively high per-iteration cost. To overcome this difficulty, we develop a long-horizon supervision mechanism based on which the solver-selection problem is cast as a prediction task. Our model is size-agnostic as we represent each optimization state with a fixed-dimensional feature vector, allowing a single shared model to generalize across problem scales without scale-specific retraining. We also establish theoretical guarantees for when dynamic solver selection outperforms static single-solver execution. We further provide a sufficient condition under which the learned selection function achieves lower expected runtime than static single-solver baselines.

Our main contributions are summarized as follows:

- **Sequential solver selection with a size-agnostic selection function**. We formulate solver selection as a sequential decision process and learn a state-dependent selection function that chooses between a cheap solver (ADMM) and a heavy solver (PD-IPM) along a single optimization trajectory. The selection function operates on problem-size-independent state features, allowing a single shared model to generalize across problem scales and structures.
- **Long-horizon supervision and low-latency execution**. We train the selection function with paired cheap and heavy rollouts from identical states, yielding reachability and remaining-runtime labels, together with a ranking loss and dataset aggregation for sample-efficient learning. To prevent inference latency from eroding wall-clock gains, we develop an asynchronous speculative execution scheme that overlaps selection inference with cheap-solver progress and replacing speculative work only when heavy solver is selected.
- **Theoretical guarantees for solver selection gains**. We establish a critical condition number that determines local solver preference and show that trajectories crossing heterogeneous condition-number regimes induce a positive solver-selection margin. We further decompose selection errors under distribution shift into runtime regret and provide a sufficient condition under which the learned selection function achieves lower expected runtime than both static single-solver baselines.

2 Related Work

2.1 Learning-Enhanced Optimization

Recent learning-to-optimize methods improve numerical optimization by embedding learned, data-driven components into classical iterative algorithms. Typical examples include algorithm unrolling: each iteration of a classical optimization method is regarded as one layer of the neural network, and multiple layers constitute an end-to-end network, with algorithm hyperparameters or subproblem representations set as trainable components. Take ADMM as an example, unrolled ADMM introduce trainable components into ADMM iterations for inverse problems and constrained optimization (An et al., 2024; Nguyen et al., 2026; Tran et al., 2026; Yang et al., 2016), while PDHG-unrolled methods have been developed for large-scale linear programming (Li et al., 2024). Related feasibility-aware learning-to-optimize frameworks further integrate constraint satisfaction into learned optimization models (Zhang & Sun, 2026; Heaton et al., 2023). Despite their effectiveness, these methods are primarily designed to improve the behavior of a single optimization algorithm. Their learned components determine how a given solver updates its variables, but they do not address which solver should be selected during execution (Ichnowski et al., 2021).

2.2 Dynamic Algorithm Selection and Solver Routing

Algorithm selection studies how to identify effective solvers from a portfolio of candidate algorithms for each given problem instance (Rice, 1976; Xu et al., 2008). Recent dynamic selection approaches

extend this idea by adapting solver configurations or switching strategies during execution based on observed problem states (Baudiš & Pošík, 2014; Adriaensen et al., 2022; Antipov & Doerr, 2026). Learning-based switching methods have also been explored for optimization problems, where decisions are adjusted using reinforcement learning or online performance feedback (Getzelman & Balaprakash, 2021; Guo et al., 2024). However, existing dynamic selection approaches often rely on short-term performance indicators, such as immediate objective improvement or residual reduction (Fialho et al., 2010), which may overlook the long-term effect of solver decisions.

3 Problem Formulation

3.1 Solving Optimization Problems via Solver Selection

We consider the following constrained optimization problem:

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \quad \text{s.t. } \boldsymbol{x} \in \Omega_i,\ 1 \le i \le I, \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the objective function, $\boldsymbol{x} \in \mathbb{R}^d$ is the optimization variable, and Ω_i is the ith constraint set. We assume that at most T iterations can be executed for solving this problem. Let $\mathcal{A} \triangleq \{A_1, \ldots, A_M\}$ denote the set of available solvers for problem (1). ***Our goal is to learn a selection function*** $\pi(s_t) : s_t \to \mathcal{A}$, $1 \le t \le T$, ***that automatically determines which solver should be employed at the*** tth ***iteration in order to minimize the runtime consumption***, where s_t is the state vector containing information necessary for solver selection. In this paper, we consider a two-solver scenario, namely, $\mathcal{A} \triangleq \{\mathcal{F}, \mathcal{S}\}$, where $\mathcal{F}$ is selected as ADMM and $\mathcal{S}$ is instantiated as the PD-IPM method. Notably, ADMM is per-iteration efficient but may require a large number of iterations to converge, whereas PD-IPM is per-iteration expensive but highly efficient in terms of the total number of iterations. It is well-known that these two methods favor different problem landscapes, and thus alternating between them according to the local landscape may significantly enhance computational efficiency. Although this paper focuses on the two-solver case, it can be readily verified that our approach can be extended to the multi-solver case. Also, note that while we assume for introductory convenience that solver selection is performed at every iteration, in practice, it is possible that the selection is carried out every several iterations.

3.2 State Information Construction

Existing learning-to-optimize methods are sensitive to the size of the optimization problem. To avoid this issue, our state vector s_t is designed to be a problem-size-independent vector which has a fixed dimension, that is,

$$s_t \triangleq \left[\log(r_t^{\mathrm{pri}} + \varepsilon_0), \log(r_t^{\mathrm{dual}} + \varepsilon_0), \rho_t^{\mathrm{pri}}, \rho_t^{\mathrm{dual}}, \log(1 + \Delta_t^S), b_t^S, t/T, f_{t,10}^S\right]^\top, \tag{2}$$

where r_t^{pri} (resp. r_t^{dual}) is the primal (resp. dual) residual, ϵ_0 is a small positive constant, $\rho_t^{\mathrm{pri}} = r_t^{\mathrm{pri}} / \max(r_{t-1}^{\mathrm{pri}}, \varepsilon_0)$ (resp. $\rho_t^{\mathrm{dual}} = r_t^{\mathrm{dual}} / \max(r_{t-1}^{\mathrm{dual}}, \varepsilon_0)$) is the ratio between two consecutive r_t^{pri}s (resp. r_t^{dual}s), Δ_t^S is the number of decisions counted from the most recent $\mathcal{S}$-action, $b_t^S = N_t^S/T$ is the ratio between the number of $\mathcal{S}$ actions already executed before the decision and the budget T, and $f_{t,10}^S$ is the fraction of $\mathcal{S}$ actions in the most 10 recent decisions. The first four components in s_t characterize the current optimization accuracy and convergence trend, while the remaining components describe the temporal context and historical usage of $\mathcal{S}$ actions. The primal and dual residuals characterize the current convergence status, while their relative changes reflect recent convergence trends and indicate whether the solver is still making effective progress. The remaining quantities describe the optimization stage and the usage of the $\mathcal{S}$ action, providing additional temporal information to distinguish states with similar residual levels but different solver-selection requirements. Also let z_t denote the stack of primal and dual variables, namely, $z_t \triangleq (x_t, \lambda_t, \mu_t)$, where x_t is the primal variable, and λ_t (resp. μ_t) represents the dual variable associated with the equality (resp. inequality) constraint. To determine whether the optimization process has reached the convergence criterion, let the convergence set be defined as

$$\mathcal{G}_\epsilon = \{z :\ e_{\mathrm{obj}}(z) \le \epsilon_{\mathrm{obj}}, v_{\mathrm{fea}}(z) \le \epsilon_{\mathrm{fea}}, r_{\mathrm{KKT}}(z) \le \epsilon_{\mathrm{KKT}}\}, \tag{3}$$

where $e_{\mathrm{obj}}(z)$, $v_{\mathrm{fea}}(z)$, and $r_{\mathrm{KKT}}(z)$ represent the objective error, feasibility violation, and KKT residual, respectively, and ϵ_{obj}, ϵ_{fea}, and ϵ_{KKT} are the corresponding convergence thresholds. Clearly, $z_t \in \mathcal{G}_\epsilon$ implies that the optimization process has met the convergence criterion.

3.3 Solver Selection as an Optimization Problem

Learning the optimal selection function can be achieved by solving the following problem:

$$\min_\theta J(\pi_\theta) \triangleq \mathbb{E}\big[\sum_{t=1}^{\min\{\tilde{T}_\epsilon(\pi_\theta),T\}} C_t(a_t) + M \cdot \mathbf{1}\{\tilde{T}_\epsilon(\pi_\theta) > T\}\big], \tag{4}$$

where θ is the parameter of π, $C_t(\cdot)$ is the time consumption of iteration t, $a_t = \pi_\theta(s_t)$ is the solver selected by π for the tth iteration, $\mathbf{1}\{\cdot\}$ is an indicator function whose value equals 1 (resp. 0) if the condition is satisfied (resp. unsatisfied), M is a tradeoff parameter, and

$$\tilde{T}_\epsilon(\pi_\theta) \triangleq \begin{cases} \min\{t \in \{1,\dots,T\} : z_t \in \mathcal{G}_\epsilon\}, & \text{if such } t \text{ exists,} \\ +\infty, & \text{otherwise.} \end{cases} \tag{5}$$

Notably, the quantity $\tilde{T}_\epsilon(\pi_\theta)$ is the index of the first iteration at which the optimization process meets the convergence criterion. Besides, $\tilde{T}_\epsilon(\pi_\theta)$ is set to $+\infty$ if the convergence criterion is not met within the overall T iterations. In (4), minimizing the first term encourages computationally efficient policies. However, optimizing runtime loss alone tends to yield a policy that is low-cost but comes at the expense of convergence, often causing the resulting optimization trajectory fail to converge. To avoid this, the indicator function is included to discourage selection results that do not reach the convergence criterion.

4 A Supervised Learning Framework for Solver Selection

In practice, directly solving problem (4) is challenging due to the implicit relation between C_t and a_t (resp. $\tilde{T}_\epsilon$ and π_θ) as well as the long-term coupling between C_ts. In other words, these functions are in nature black-box. For this reason, traditional optimization methods are nonapplicable. To address this problem, we develop a supervised learning framework that transforms solver selection into a prediction task. On this basis, by incorporating a dataset aggregation mechanism, our approach can identify a high-quality selection function with only a few labeled training data.

4.1 Solver Selection via Learning Predictions

Consider a set of training instances of problem (1). Suppose we can enumerate all the combinations of solver selections across T iterations for each instance. Our idea is to train a neural network to predict whether the optimization process will converge as well as to estimate the remaining runtime, given an arbitrary solver choice (e.g., selecting either $\mathcal{F}$ or $\mathcal{S}$) at an arbitrary state s_t. With such a neural network, optimal solver selection can be realized by comparing the predicted results associated with different solvers at every iteration.

To train a neural network for reachability and remaining runtime prediction, let $y_t^{\text{reach}}(s_t, a_t)$ denote the reach indicator, whose value equals 1 (resp. 0) if selecting solver a_t at state s_t eventually reaches (resp. fails to reach) the convergence criterion within the remaining iteration budget. Also let $y_t^{\text{time}}(s_t, a_t)$ denote the runtime consumption (from iteration t until convergence or the iteration budget is exhausted) associated with the selected solver a_t. Since unsuccessful solver choices should be discouraged, we define the time target as

$$y_t^{\text{time}}(s_t, a_t) = \log\left(1 + \bar{T}(s_t, a_t) + \lambda_{\text{fail}}(1 - y_t^{\text{reach}}(s_t, a_t))\right), \tag{6}$$

where $\bar{T}(s_t, a_t)$ is the consumed runtime (normalized by the maximum runtime), and λ_{fail} is a penalty parameter for solver selections that fail to reach the convergence criterion. The concatenated vector $(s_t, a_t, y_t^{\text{reach}}(s_t, a_t), y_t^{\text{time}}(s_t, a_t))$ constitute a training sample. The learning objective combines reach prediction, runtime prediction, and solver ranking:

$$\mathcal{L}(\theta) = \lambda_{\text{reach}}\mathcal{L}_{\text{reach}} + \lambda_{\text{time}}\mathcal{L}_{\text{time}} + \lambda_{\text{rank}}\mathcal{L}_{\text{rank}}. \tag{7}$$

where λ_{reach}, λ_{time}, λ_{rank} are tradeoff parameters. Since reach prediction is a binary prediction task, $\mathcal{L}_{\text{reach}}$ is chosen as a binary cross-entropy loss defined as

$$\mathcal{L}_{\text{reach}} = -\tfrac{1}{|\mathcal{D}|}\sum_{\mathcal{D}}\sum_{t=1}^{T}\sum_{a\in\mathcal{A}}\left[y_t^{\text{reach}}\log\hat{p}_\theta^{\text{reach}}(s_t,a) + (1-y_t^{\text{reach}})\log\big(1-\hat{p}_\theta^{\text{reach}}(s_t,a)\big)\right]. \tag{8}$$

where $(\hat{p}_\theta^{\text{reach}}(s_t, a_t), \hat{y}_\theta^{\text{time}}(s_t, a_t))$ is the output of the neural network, $\hat{p}_\theta^{\text{reach}}(s_t, a_t)$ is the prediction of the probability of reaching the convergence criterion after selecting solver a_t, $\hat{y}_\theta^{\text{time}}(s_t, a_t)$ is the prediction of the remaining runtime, $\mathcal{D}$ is the set of training instances of problem (1). Whereas $\mathcal{L}_{\text{time}}$ is chosen as the Huber loss:

$$\mathcal{L}_{\text{time}} = \frac{1}{|\mathcal{D}|} \sum_{\mathcal{D}} \sum_{1 \le t \le T} \sum_{a_t \in \mathcal{A}} \text{Huber}_{0.5} \left(\hat{y}_\theta^{\text{time}}(s_t, a_t) - y_t^{\text{time}} \right). \tag{9}$$

In the inference stage, given a state s_t, we choose the solver with a smaller predicted runtime $\hat{y}_\theta^{\text{time}}(s_t, a_t)$, since the time target already incorporates the failure penalty. This means that the relative ordering of the predicted runtime, rather than the absolute value of the predictions, determines the solver choice. Therefore, we introduce a ranking loss to encourage the model to preserve the correct ordering. For the two-solver case, define $y_t^{\text{rank}} = \mathbf{1}\{y_t^{\text{time}}(s_t, \mathcal{F}) < y_t^{\text{time}}(s_t, \mathcal{S})\}$. The predicted time difference is $d_{\theta,t} = \hat{y}_\theta^{\text{time}}(s_t, \mathcal{S}) - \hat{y}_\theta^{\text{time}}(s_t, \mathcal{F})$. The ranking loss is then defined as

$$\mathcal{L}_{\text{rank}} = -\frac{1}{|\mathcal{D}|T} \sum_{\mathcal{D}} \sum_{1 \le t \le T} \left[y_t^{\text{rank}} \log \sigma(d_{\theta,t}) + (1 - y_t^{\text{rank}}) \log(1 - \sigma(d_{\theta,t})) \right], \tag{10}$$

where $\sigma(\cdot)$ denotes the sigmoid function, which smoothly maps the signed runtime difference $d_{\theta,t}$ into a relative solver preference while preserving the ordering implied by its sign.

4.2 Sample-Efficient Training via Dataset Aggregation

The above supervised learning framework requires labeled training data. However, obtaining the label information $(y_t^{\text{reach}}(s_t, a_t), y_t^{\text{time}}(s_t, a_t))$ is expensive because they can only be calculated after finishing the complete optimization process. Besides, the number of all possible solver combinations can be gigantically large, making it impractical to enumerate all training data. To address this problem, we employ a dataset aggregation mechanism to realize sample-efficient training.

For each problem instance in $\mathcal{D}$, we first generate a small training data set $\mathcal{T}_0$. In this set, the ith sample is obtained by sequentially performing i cheap iterations and $T - i$ heavy iterations. We then train a coarse neural network θ_0 based on $\mathcal{T}_0$. The trained neural network is then executed on each problem instances in $\mathcal{D}$ to generate a new optimization trajectory. These trajectories contribute an incremental training set $\Delta\mathcal{T}_0$. Hence the training set for the next stage is constructed as $\mathcal{T}_1 = \mathcal{T}_0 \cup \Delta\mathcal{T}_0$. To improve the diversity of $\mathcal{T}_1$, the exploration trick is employed. Specifically, for each solver selection made by the trained neural network, the selected solver has a probability of ϵ to be switched to the other one. Notably, duplicated labels are removed during dataset aggregation to avoid redundant supervision. When conflicting labels occur before and after network update, the labels generated by the most recent network update are retained.

4.3 Latency Reduction via Asynchronous Inference Design

Solver selection introduces additional inference latency, as the optimization process must wait for the neural network output before proceeding. To reduce this latency, we propose an asynchronous inference execution mechanism. The main idea is to decouple the neural network computation from the solver execution. While the neural network computation consumes runtime, the optimization process advances with the cheap solver, eliminating the need to wait for inference. If the neural network output is exactly the cheap solver, then the progress already made can be directly inherited, otherwise the heavy solver is reloaded.

5 Condition-Number Regimes and Solver Tradeoffs

We analyze when dynamic solver selection is beneficial under a local additive-runtime approximation. Focusing on the local KKT condition number, we derive a critical condition number that determines which solver is favorable and show that heterogeneous condition-number regimes create runtime complementarity between cheap and heavy solvers, yielding a positive solver-selection margin along trajectories that cross distinct regimes.

Local KKT Geometry and Runtime Model. Recall that $z_t = (x_t, \lambda_t, \mu_t) \in \mathbb{R}^{\tilde{d}}$ contains the primal and dual variables. Let $r_t \triangleq r_{\text{KKT}}(z_t)$ denote the KKT residual at iteration t. Within a local region around z_t, let $\text{Hess}(z_t)$ denote the Hessian of the Lagrangian evaluated at z_t, and also assume $m(z_t)I \preceq \text{Hess}(z_t) \preceq L_{\text{sp}}(z_t)I$, where $0 < m(z_t) \le L_{\text{sp}}(z_t) < \infty$. The local condition number

is defined as $\kappa(z_t) \triangleq L_{\mathrm{sp}}(z_t)/m(z_t) \geq 1$. Let $L_H(z_t)$ denote the local Lipschitz constant of the Lagrangian Hessian, and define $\eta(z_t) \triangleq 2m(z_t)^2/L_H(z_t)$. The cheap solver $\mathcal{F}$ (i.e., ADMM) has per-iteration runtime $C_t(\mathcal{F}) = c_f \cdot \tilde{d}$, whereas the heavy solver $\mathcal{S}$ (i.e., PD-IPM) has per-iteration runtime $C_t(\mathcal{S}) = c_s \cdot \tilde{d}^p$, where $c_f, c_s > 0$ are constants and $p \in [2, 3]$. These assumptions are reasonable, since according to (Boyd et al., 2011; Deng & Yin, 2016; Wright, 1997), $\mathcal{F}$ (resp. $\mathcal{S}$) is linearly (resp. quadratically) convergent under mild conditions. The following lemma highlights the distinct local convergence behaviors of the two solvers in terms of the number of required iterations.

Lemma 1 (Iteration Complexity)**.** *Assume that the ADMM (resp. PD-IPM) solver is linearly (resp. quadratically) convergent, then at z_t, the estimated numbers of iterations required by ADMM and PD-IPM to converge to a prescribed tolerance ϵ_{KKT} are respectively given by*

$$N_{\mathcal{F}}^{\mathrm{sur}}(z_t) \triangleq \tfrac{\kappa(z_t)}{c_0} \cdot \ln\left(\tfrac{r_t}{\epsilon_{\mathrm{KKT}}}\right),\ N_{\mathcal{S}}^{\mathrm{sur}}(z_t) \triangleq \log_2 \max\left\{2,\ \log_2\left(\max\left\{2,\ \tfrac{\eta(z_t)}{\epsilon_{\mathrm{KKT}}}\right\}\right)\right\}. \tag{11}$$

The proof is given in Appendix A.1.

Critical Condition-Number Partitioning. Let $B_t \triangleq T - t + 1$ denote the remaining iteration budget. Define the continuous runtime-difference surrogate as

$$D(z_t) \triangleq C_t(\mathcal{S}) \cdot N_{\mathcal{S}}^{\mathrm{sur}}(z_t) - C_t(\mathcal{F}) \cdot N_{\mathcal{F}}^{\mathrm{sur}}(z_t), \tag{12}$$

where $D(z_t) > 0$ (resp. $D(z_t) < 0$) implies that $\mathcal{F}$ (resp. $\mathcal{S}$) is more favorable in terms of predicted runtime.

Theorem 1 (Critical Condition Number)**.** *Suppose that both solvers can reach the prescribed tolerance within the remaining iteration budget, i.e., $\max\{N_{\mathcal{F}}^{\mathrm{sur}}(z_t), N_{\mathcal{S}}^{\mathrm{sur}}(z_t)\} \leq B_t$, and that $r_t \leq \eta(z_t)/2$. Then $\mathcal{S}$ requires a smaller predicted runtime (i.e., $D(z_t) < 0$) if*

$$\kappa(z_t) > \kappa_{\mathrm{crit}}(z_t) \triangleq \tfrac{c_0 c_s d^{p-1} N_{\mathcal{S}}^{\mathrm{sur}}(z_t)}{c_f \ln(r_t/\epsilon_{\mathrm{KKT}})}. \tag{13}$$

Conversely, $\mathcal{F}$ necessitates a smaller predicted runtime if $\kappa(z_t) < \kappa_{\mathrm{crit}}(z_t)$.

The proof is given in Appendix A.2. Theorem 1 identifies the critical condition number that determines which solver is more advantageous in terms of the remaining runtime. Explicitly evaluating $\kappa(z_t)$ is expensive. This motivates learning the solver-selection function $\pi_\theta(s_t)$ from the problem-size-independent state information s_t, allowing the model to predict solver preference without explicitly resorting to the local condition number.

Certified Runtime Advantage from Regime Crossing. Consider a sequence of $\{z_t\}_{t=1}^K$, with $K \leq T$. We adopt an additive runtime model in which solver selection primarily affects the runtime incurred at each state without substantially changing the macro-level optimization trajectory. Let $\tau_a(z_t)$ denote the runtime incurred by selecting solver $a \in \{\mathcal{F}, \mathcal{S}\}$ at state z_t, and define $\mathcal{T}(\pi) \triangleq \sum_{t=1}^K \tau_{\pi(s_t)}(z_t)$. The oracle selection rule is $\pi^*(z_t) \triangleq \arg\min_{a \in \{\mathcal{F},\mathcal{S}\}} \tau_a(z_t)$, with runtime $\mathcal{T}^* \triangleq \sum_{t=1}^K \min_{a \in \{\mathcal{F},\mathcal{S}\}} \tau_a(z_t)$. Let $\pi_{\mathcal{F}}$ and $\pi_{\mathcal{S}}$ denote the static selection functions that always select $\mathcal{F}$ and $\mathcal{S}$, respectively. The instance-level solver-selection margin is defined as

$$\Gamma(\xi) \triangleq \min\{\mathcal{T}(\pi_{\mathcal{F}}), \mathcal{T}(\pi_{\mathcal{S}})\} - \mathcal{T}^*, \tag{14}$$

Let $\mathcal{P}$ denote the population distribution over problem instances (e.g. $\mathcal{D}$ is a subset of $\mathcal{P}$). The distribution-level margin is defined as $\Gamma_{\mathcal{P}} \triangleq \mathbb{E}_{\xi \sim \mathcal{P}}[\Gamma(\xi)]$. Let $G_{\mathcal{F}}(\xi)$ and $G_{\mathcal{S}}(\xi)$ denote the certified excess runtime of the two static selection functions, constructed using the certified region and the corresponding runtime-surrogate error bounds. Their formal definitions are given in Appendix A.3. The following theorem shows that different local conditions can create different runtime advantages for $\mathcal{F}$ and $\mathcal{S}$ along the same optimization trajectory, making static single-solver suboptimal.

Theorem 2 (Certified Runtime Advantage from Regime Crossing)**.** *For any instance ξ of problem (1), the solver-selection margin satisfies*

$$\Gamma(\xi) \geq \min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\}. \tag{15}$$

If the optimization trajectory visits at least one certified state favoring $\mathcal{F}$ (resp. $\mathcal{S}$) beyond the surrogate-error tolerance, then both static selection functions incur strictly larger runtime than the oracle selection rule, and hence $\Gamma(\xi) > 0$. Furthermore, it holds

$$\Gamma_{\mathcal{P}} \geq \mathbb{E}_{\xi \sim \mathcal{P}}\left[\min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\}\right]. \tag{16}$$

If certified regime crossing occurs with positive probability under $\xi \sim \mathcal{P}$, then $\Gamma_{\mathcal{P}} > 0$.

The proof is given in Appendix A.3.

6 Theoretical Guarantees for Learned Solver Selection

Having established that heterogeneous problem landscapes can yield a positive solver-selection margin, we now study when the learned selection function π_θ can preserve this advantage and achieve lower expected runtime than static single-solver execution. We first relate solver-selection errors to runtime regret, then connect this regret to the ranking objective used for training, and finally combine the resulting learning error with the solver-selection margin.

Under the additive runtime model in Section 5, define the runtime difference as $\Delta\tau(z_t) \triangleq \tau_{\mathcal{S}}(z_t) - \tau_{\mathcal{F}}(z_t)$, and assume that $\bar{\Delta} \triangleq \sup_z |\Delta\tau(z)| < \infty$. Thus, $\Delta\tau(z_t) > 0$ implies that $\mathcal{F}$ is more favorable in terms of the remaining runtime. The oracle selection rule π^* selects $\mathcal{F}$ (resp. $\mathcal{S}$) when $\Delta\tau(z_t) > 0$ (resp. otherwise).

Lemma 2 (Exact Runtime Regret Decomposition). *Under the additive runtime model, the runtime regret of any solver-selection function π relative to the oracle runtime $\mathcal{T}^*$ satisfies*

$$\mathcal{T}(\pi) - \mathcal{T}^* = \textstyle\sum_{t=1}^{K} |\Delta\tau(z_t)| \cdot \mathbf{1}\{\pi(s_t) \neq \pi^*(z_t)\}. \tag{17}$$

The proof is given in Appendix A.4. Lemma 2 shows that solver-selection mistakes translate directly into runtime regret, weighted by the runtime difference between the two solvers. We next relate such mistakes to the ranking target used to train the selection function.

The supervised learning framework defines the ranking target $y_t^{\mathrm{rank}} = \mathbf{1}\{y_t^{\mathrm{time}}(s_t, \mathcal{F}) < y_t^{\mathrm{time}}(s_t, \mathcal{S})\}$. Define $q_t \triangleq 2y_t^{\mathrm{rank}} - 1 \in \{-1, +1\}$, where $q_t = +1$ (resp. $q_t = -1$) favors $\mathcal{F}$ (resp. $\mathcal{S}$). Recall that $d_{\theta,t} \triangleq \widehat{y}_\theta^{\mathrm{time}}(s_t, \mathcal{S}) - \widehat{y}_\theta^{\mathrm{time}}(s_t, \mathcal{F})$, so $\pi_\theta(s_t)$ selects $\mathcal{F}$ (resp. $\mathcal{S}$) if $d_{\theta,t} > 0$ (resp. otherwise). Define the oracle preference label $q_t^* \triangleq 2 \cdot \mathbf{1}\{\Delta\tau(z_t) > 0\} - 1$ and the expected runtime regret $\mathcal{E}_{\mathcal{P}}(\pi_\theta) \triangleq \mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_\theta) - \mathcal{T}^*]$. The regret can be decomposed as

$$\mathcal{E}_{\mathcal{P}}(\pi_\theta) \leq \mathcal{E}_{\mathrm{learn}}(\pi_\theta) + \mathcal{E}_{\mathrm{target}}, \tag{18}$$

where $\mathcal{E}_{\mathrm{learn}}(\pi_\theta)$ measures solver-ordering errors relative to the supervised ranking target, while $\mathcal{E}_{\mathrm{target}}$ captures the mismatch between the supervised target and the runtime-oracle preference. Their formal definitions and the derivation of (18) are provided in Appendix A.5.

The ranking objective is optimized on states represented in the training data, whereas the learned selection function is evaluated on states encountered during inference. We therefore account for the resulting distribution shift. To achieve this, we make the following assumption.

Assumption 1 (State-Distribution Concentrability and Generalization Gap). *Let $Z_{\pi_\theta}(s)$ denote the average state distribution induced by π_θ during inference, and let $Z_{\mathrm{data}}(s)$ denote the state distribution represented by the aggregated training data. The inference-to-training distribution shift satisfies $Z_{\pi_\theta}(s) \leq \beta \cdot Z_{\mathrm{data}}(s)$ for all s, where $\beta \triangleq \sup_s Z_{\pi_\theta}(s)/Z_{\mathrm{data}}(s) < \infty$. Furthermore, let $\mathcal{L}_{\mathrm{rank}}^{\mathrm{pop}}(\theta)$ and $\mathcal{L}_{\mathrm{rank}}^{\mathrm{train}}(\theta)$ denote the population and empirical ranking losses, respectively. We assume that $\mathcal{L}_{\mathrm{rank}}^{\mathrm{pop}}(\theta) \leq \mathcal{L}_{\mathrm{rank}}^{\mathrm{train}}(\theta) + \epsilon_{\mathrm{gen}}$, where $\epsilon_{\mathrm{gen}} \geq 0$ denotes the generalization gap.*

The dataset aggregation mechanism expands training-state coverage toward states encountered by the learned selection function during inference, reducing the distribution shift captured by β.

Lemma 3 (Learning Error Bound of Ranking Loss). *Under Assumption 1, it holds*

$$\mathcal{E}_{\mathrm{learn}}(\pi_\theta) \leq (\bar{\Delta} K \beta / \ln 2) \left(\mathcal{L}_{\mathrm{rank}}^{\mathrm{train}}(\theta) + \epsilon_{\mathrm{gen}}\right). \tag{19}$$

The proof is given in Appendix A.6. Lemma 3 shows that the ranking loss controls solver-selection error, up to the distribution-shift and generalization factors.

Finally, we combine this learning-error bound with the positive solver-selection margin established in Section 5. Let $\underline{\Gamma}_{\mathcal{P}} \triangleq \mathbb{E}_{\xi\sim\mathcal{P}}[\min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\}]$ denote the certified lower bound on the solver-selection margin from Theorem 2. The following theorem shows that the learned solver-selection function preserves the runtime advantage when the available solver-selection margin is sufficiently large relative to the target mismatch and learning error.

Theorem 3 (Sufficient Condition for Learned Solver-Selection Improvement). *Suppose that $\underline{\Gamma}_{\mathcal{P}} > \mathcal{E}_{\mathrm{target}}$. Then π_θ achieves a smaller expected runtime than all-$\mathcal{F}$ and all-$\mathcal{S}$ selections, namely,*

$$\mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_\theta)] < \min\left\{\mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_{\mathcal{F}})], \mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_{\mathcal{S}})]\right\}, \tag{20}$$

Table 1: ACOPF evaluation results.

Generated validation operating points

Case	Reach (%) Heavy	Cheap	GridSFM	ML4OPF	Ours	Runtime (s) Heavy	Cheap	GridSFM	ML4OPF	Ours
case300_ieee	47	0	63	65	**100**	6.024	0.760	3.399	2.962	**0.516**
case500_goc	89	0	96	95	**100**	8.051	1.098	2.216	1.867	**0.655**
case588_sdet	**100**	0	**100**	99	**100**	2.545	1.326	3.888	**1.332**	1.520
case793_goc	**100**	0	**100**	**100**	**100**	1.313	1.420	2.711	**1.031**	1.039
All	84.0	0	89.75	89.75	**100**	4.483	1.151	3.053	1.798	**0.932**

Original PGLib cases

Case	Reach (%) Heavy	Cheap	GridSFM	ML4OPF	Ours	Runtime (s) Heavy	Cheap	GridSFM	ML4OPF	Ours
case300_ieee	**100**	0	**100**	**100**	**100**	2.980	0.782	0.744	0.731	**0.507**
case500_goc	**100**	0	**100**	**100**	**100**	7.760	1.117	1.505	1.089	**0.712**
case588_sdet	**100**	0	**100**	**100**	**100**	2.259	1.344	2.710	**1.009**	1.371
case793_goc	**100**	0	**100**	**100**	**100**	1.322	1.438	3.937	0.992	**0.975**
Pooled	**100**	0	**100**	**100**	**100**	3.580	1.170	2.224	0.955	**0.891**

provided that

$$\mathcal{L}_{\text{rank}}^{\text{train}}(\theta) < (\ln 2/\bar{\Delta}K\beta)\,(\underline{\Gamma}_{\mathcal{P}} - \mathcal{E}_{\text{target}}) - \epsilon_{\text{gen}}. \tag{21}$$

The proof is given in Appendix A.7.

7 Experimental Evaluation

We evaluate our method on two complementary classes of optimization problems: non-convex AC Optimal Power Flow (ACOPF) (Babaeinejadsarookolaee et al., 2019) and convex Quadratic Programming (QP) (Maros & Mészáros, 1999). Our experiments examine whether the proposed approach can reduce the end-to-end wall-clock time while maintaining convergence reliability, and whether a single shared model can generalize across different problem structures and scales. Detailed solver implementations and training setting are provided in Appendix B. We further report experiments on zero-shot generalization to unseen problem scales and structures in Appendix C.2 and robustness to alternative Cheap–Heavy configurations in Appendix C.3.

7.1 Non-Convex Benchmark: AC Optimal Power Flow

We first consider ACOPF. We construct the benchmark from four PGLib network topologies: case300_ieee, case500_goc, case588_sdet, and case793_goc. For each topology, we generate 1,000 problem instances. The convergence criterion is set as $\max\{e_{\text{obj}}, e_{\text{feas}}, e_{\text{KKT}}\} \leq 10^{-4}$. We compare against the conventional heavy solver and representative learning-based ACOPF approaches, including GridSFM (Yang et al., 2026) and a PCA-based Compact Learning baseline (Park et al., 2023) implemented within the ML4OPF framework. Detailed instance generation, numerical implementations, and baseline configurations are provided in Appendix B.1. For our approach, a single neural network is trained across for all four problem scales, whereas either GridSFM or ML4OPF are separately trained for each problem scale.

As shown in Table 1, our method achieves full convergence reach on both the generated validation instances and the original PGLib cases, while attaining the lowest overall average runtime among the compared methods. In particular, our selection scheme consistently improves the efficiency–reliability tradeoff over the static solvers and the learning-based baselines with different network scales. These results demonstrate that our method generalizes well across scale-varying ACOPF instances without requiring scale-specific model training.

7.2 Convex Benchmark: Parametric Quadratic Programming

We next consider convex parametric quadratic programming (QP), where global optimality can be characterized through the KKT conditions. Nevertheless, the computational effort required to reach a prescribed numerical accuracy can still vary substantially across solvers. We generate 5,000 QP instances from 5 specific instance in the Maros-Mészáros benchmark (Maros & Mészáros, 1999):

Table 2: Maros-Mészáros QP evaluation result.

Generated joint-parametric test set												
	Reach (%)						Runtime (ms)					
Case	Heavy	Cheap	GNN-QP	GNN-DAQP	LAH-Accel	Ours	Heavy	Cheap	GNN-QP	GNN-DAQP	LAH-Accel	Ours
PRIMAL1	100	0	100	100	100	100	13.26	19.07	13.32	**9.68**	12.28	12.10
PRIMAL2	100	100	100	100	100	100	21.14	24.45	22.04	32.35	26.89	**19.30**
PRIMAL3	100	100	100	100	100	100	56.64	**47.88**	57.21	60.69	86.55	54.33
PRIMALC5	100	0	100	100	100	100	**15.99**	21.99	16.11	29.05	20.06	16.37
PRIMALC8	100	0	100	100	100	100	33.25	41.22	36.65	165.10	52.77	**30.05**
All	**100**	40	**100**	**100**	**100**	**100**	28.06	30.92	29.07	59.37	39.71	**26.43**
Original Maros-Mészáros QP cases												
	Reach (%)						Runtime (ms)					
Case	Heavy	Cheap	GNN-QP	GNN-DAQP	LAH-Accel	Ours	Heavy	Cheap	GNN-QP	GNN-DAQP	LAH-Accel	Ours
PRIMAL1	100	0	100	100	100	100	18.73	19.34	16.71	15.84	16.25	**12.91**
PRIMAL2	100	100	100	100	100	100	29.79	24.91	29.93	33.65	36.14	**19.09**
PRIMAL3	100	100	100	100	100	100	68.25	**46.53**	68.84	98.15	99.72	56.19
PRIMALC5	100	0	100	100	100	100	18.57	21.87	16.83	**10.52**	19.76	16.38
PRIMALC8	100	0	100	100	100	100	39.08	41.13	40.20	58.02	70.59	**26.39**
Pooled	**100**	40	**100**	**100**	**100**	**100**	34.88	30.76	34.50	43.24	48.49	**26.19**

Table 3: Runtime comparison of synchronous and asynchronous inference(ms).

	QP						ACOPF					
	Generated			Original			Generated			Original		
	Sync	Async	Speedup	Sync	Async	Speedup	Sync	Async	Speedup	Sync	Async	Speedup
Runtime	32.92	**25.36**	**1.30**×	33.09	**24.99**	**1.32**×	1001	**929**	**1.08**×	948	**873**	**1.09**×

PRIMAL1, PRIMAL2, PRIMAL3, PRIMALC5, and PRIMALC8, with the number of optimization variables ranging from 287 to 745. Among all instances, 4,000 instances are used for training and 1,000 for evaluation. The convergence criterion is set as $\max\{e_{\text{obj}}, e_{\text{feas}}, e_{\text{KKT}}\} \leq 10^{-6}$. We compare against the all-Heavy solver as well as representative learning-to-optimize baselines, including direct GNN-based solution prediction (GNN-QP) (Chen et al., 2025), GNN-based active-set warm starting (GNN-DAQP) (Schmidtobreick et al., 2026), and learned algorithm hyperparameters (LAH-Accel) (Sambharya & Stellato, 2026). For our approach, we train a single neural network for all problem scales, whereas GNN-QP, GNN-DAQP, and LAH-Accel should train multiple neural networks, each for one problem scale. We also evaluate the performance of respective methods on the original Maros–Mészáros QP instances. Detailed instance generation, reference solution construction, numerical implementations, and baseline configurations are provided in Appendix B.2.

As shown in Table 2, our method achieves full convergence reach while attaining the lowest overall average runtime on both the generated test set and the original QP instances. Upon only a single neural network, our method consistently provides a favorable efficiency–reliability tradeoff compared to other baselines. These results demonstrate effective generalization across scale-varying convex QPs without requiring scale-specific model training.

7.3 Latency Reduction via Asynchronous Inference

We compare asynchronous speculative execution with synchronous inference under identical optimization decisions and convergence outcomes. As shown in Table 3, asynchronous execution consistently reduces wall-clock runtime across evaluation settings, demonstrating that speculative execution can hide model inference latency without compromising optimization reliability.

8 Conclusion

We presented a state-dependent solver-selection framework that dynamically chooses between different solvers. A size-agnostic selection function is trained with paired cheap–heavy rollouts, with asynchronous speculative execution reducing inference latency. We derive a critical condition number and show that heterogeneous conditioning induces a positive solver-selection margin, with selection errors translating into runtime regret. On ACOPF and QP, a single shared selection function generalizes across scales, topologies, and matrix structures while reducing wall-clock runtime and preserving convergence reliability.

AI USE STATEMENT

In this work, we used generative AI to assist with code generation, language polishing, and limited manuscript drafting. We reviewed all AI-assisted work and verified the generated code before use. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, Appendix B documents the ADMM and PD-IPM implementations, numerical configurations, and solver-switching mechanisms for ACOPF and QP. Appendix B.3 provides our network architecture, feature normalization, training hyperparameters, dataset-aggregation procedure, training seeds, and checkpoint selection.

ETHICS STATEMENT

We have considered the potential ethical implications of this work, including issues related to fairness, privacy, security, and possible misuse. Our work is intended for research purposes, and we encourage responsible use of the proposed methods and careful consideration of their limitations when applied in real-world settings. We have made efforts to report our methodology and experimental results accurately and transparently, and we take responsibility for the integrity of the research presented in this paper.

REFERENCES

Steven Adriaensen, André Biedenkapp, Gresa Shala, Noor H. Awad, Theresa Eimer, Marius Lindauer, and Frank Hutter. Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research*, 75:1633–1699, 2022.

Weixin An, Yuanyuan Liu, Fanhua Shang, Hongying Liu, and Licheng Jiao. DEs-Inspired accelerated unfolded linearized ADMM networks for inverse problems. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):5319–5333, 2024.

Denis Antipov and Carola Doerr. When switching algorithms helps: A theoretical study of online algorithm selection. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1304–1312, 2026.

Sogol Babaeinejadsarookolaee, Adam Birchfield, Richard Christie, et al. The power grid library for benchmarking AC optimal power flow algorithms. Technical report, IEEE PES PGLib-OPF Task Force, 2019.

Petr Baudiš and Petr Pošík. Online black-box algorithm portfolios for continuous optimization. *International Conference on Parallel Problem Solving from Nature*, pp. 40–49, 2014.

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

Ziang Chen, Xiaohan Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. Expressive power of graph neural networks for (mixed-integer) quadratic programs. *International Conference on Machine Learning*, pp. 7880–7911, 2025.

Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66:889–916, 2016.

Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michele Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60(1): 25–64, 2010.

Grant Getzelman and Prasanna Balaprakash. Learning to switch optimizers for quadratic programming. *Asian Conference on Machine Learning*, 157:1553–1568, 2021.

Paul Goulart and Yuwen Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives. *Mathematical Programming Computation*, pp. 1–83, 2026.

Hongshu Guo, Yining Ma, Zeyuan Ma, Jiacheng Chen, Xinglin Zhang, Zhiguang Cao, et al. Deep reinforcement learning for dynamic algorithm selection: A proof-of-principle study on differential evolution. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 54(7):4247–4259, 2024.

Howard Heaton, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Safeguarded learned convex optimization. *AAAI Conference on Artificial Intelligence*, 37(6):7848–7855, 2023.

Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, et al. Accelerating quadratic optimization with reinforcement learning. *Advances in Neural Information Processing Systems*, 34:21043–21055, 2021.

Michael Klamkin, Mathieu Tanneau, and Pascal Hentenryck. Pglearn–An open-source learning toolkit for optimal power flow. *arXiv preprint arXiv:2505.22825*, 2025.

Bingheng Li, Linxin Yang, Yupeng Chen, Senmiao Wang, Haitao Mao, Qian Chen, et al. PDHG-Unrolled learning-to-optimize method for large-scale linear programming. *International Conference on Machine Learning*, 235:29164–29180, 2024.

István Maros and Csaba Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1–4):671–681, 1999.

Binh Nguyen, Trinh Tran, and Truong Nghiem. LEAF: A learning-enabled ADMM framework for accelerated convex optimization. *arXiv preprint arXiv:2606.08993*, 2026.

Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2 edition, 2006. ISBN 978-0-387-30303-1.

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

John R Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Rajiv Sambharya and Bartolomeo Stellato. Learning algorithm hyperparameters for fast parametric convex optimization. *SIAM Journal on Mathematics of Data Science*, 8(3):649–676, 2026.

Ella Schmidtobreick, Daniel Arnström, Paul Häusner, and Jens Sjölund. Warm-Starting active-set solvers using graph neural networks. *Annual Learning for Dynamics and Control Conference*, 331:662–677, 2026.

Roland Schwan, Yuning Jiang, Daniel Kuhn, and Colin Jones. PIQP: A proximal interior-point quadratic programming solver. In *IEEE Conference on Decision and Control*, pp. 1088–1093, 2023.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

Richard Tapia, Yin Zhang, and Matthew Saltzman. The Mehrotra predictor-corrector interior-point method as a perturbed composite Newton method. *SIAM Journal on Optimization*, 6(1):47–56, 1996.

Trinh Tran, Binh Nguyen, and Truong Nghiem. HUANet: Hard-constrained unrolled ADMM for constrained convex optimization. *arXiv preprint arXiv:2604.13179*, 2026.

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

Stephen J Wright. *Primal-dual interior-point methods*. SIAM, 1997.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

Weiwei Yang, Andrea Lima, Thiago Spina, Spencer Fowers, Baosen Zhang, and Chris White. GridSFM: A foundation model for AC optimal power flow. Technical Report, 2026.

Yan Yang, Jian Sun, Huibin Li, and Zongben Xu. Deep ADMM-net for compressive sensing MRI. *Advances in Neural Information Processing Systems*, 29:1–9, 2016.

Hanwen Zhang and Haijian Sun. Feasibility-aware learning-to-optimize for wireless resource allocation. *IEEE Wireless Communications*, 2026.

A Proofs and Extended Theoretical Details

A.1 Proof of Lemma 1

Preliminaries. For a fixed variable vector z_t, let $r_{t,k}^{\mathcal{F}}$ and $r_{t,k}^{\mathcal{S}}$ denote the KKT residuals after k additional iterations of $\mathcal{F}$ and $\mathcal{S}$, respectively, starting from z_t, with $r_{t,0}^{\mathcal{F}} = r_{t,0}^{\mathcal{S}} = r_t$. Following standard local convergence results deduced in (Boyd et al., 2011; Deng & Yin, 2016; Wright, 1997), we assume that the ADMM solver $\mathcal{F}$ exhibits linear convergence, namely, $r_{t,k+1}^{\mathcal{F}} \leq (1 - c_0/\kappa(z_t))r_{t,k}^{\mathcal{F}}$ for some $c_0 \in (0,1)$. Within the quadratic convergence basin $r_{t,k}^{\mathcal{S}} \leq \eta(z_t)/2$, the PD-IPM solver $\mathcal{S}$ possesses quadratic convergence, i.e., $r_{t,k+1}^{\mathcal{S}} \leq (r_{t,k}^{\mathcal{S}})^2/\eta(z_t)$, where $\eta(z_t) \triangleq 2m(z_t)^2/L_H(z_t)$ and $L_H(z_t)$ denotes the local Hessian Lipschitz constant.

Proof. For the ADMM solver $\mathcal{F}$, the linear contraction gives

$$r_{t,k}^{\mathcal{F}} \leq \left(1 - \frac{c_0}{\kappa(z_t)}\right)^k \cdot r_t \leq \exp\left(-\frac{c_0 k}{\kappa(z_t)}\right) \cdot r_t. \tag{22}$$

Thus, $r_{t,k}^{\mathcal{F}} \leq \epsilon_{\mathrm{KKT}}$ is guaranteed whenever $k \geq \frac{\kappa(z_t)}{c_0} \ln(r_t/\epsilon_{\mathrm{KKT}})$, which motivates $N_{\mathcal{F}}^{\mathrm{sur}}(z_t) \triangleq \frac{\kappa(z_t)}{c_0} \ln(r_t/\epsilon_{\mathrm{KKT}})$.

For the PD-IPM solver $\mathcal{S}$, let $y_{t,k} \triangleq r_{t,k}^{\mathcal{S}}/\eta(z_t)$. Under $r_t \leq \eta(z_t)/2$, we have $y_{t,0} \leq 1/2$, and the quadratic contraction gives $y_{t,k} \leq (y_{t,0})^{2^k} \leq (1/2)^{2^k}$. Hence, $r_{t,k}^{\mathcal{S}} \leq \epsilon_{\mathrm{KKT}}$ is guaranteed whenever $k \geq \log_2 \log_2(\eta(z_t)/\epsilon_{\mathrm{KKT}})$. Therefore, the number of iterations that guarantees PD-IPM convergence is given as

$$N_{\mathcal{S}}^{\mathrm{sur}}(z_t) \triangleq \log_2 \max\left\{2, \log_2\left(\max\left\{2, \frac{\eta(z_t)}{\epsilon_{\mathrm{KKT}}}\right\}\right)\right\}, \tag{23}$$

where the nested max operators ensure that the surrogate keeps positive and well-defined. □

A.2 Proof of Theorem 1

Proof. Recall that

$$D(z_t) \triangleq C_t(\mathcal{S}) \cdot N_{\mathcal{S}}^{\mathrm{sur}}(z_t) - C_t(\mathcal{F}) \cdot N_{\mathcal{F}}^{\mathrm{sur}}(z_t) \tag{24}$$

Substituting $C_t(\mathcal{F}) = c_f \cdot \tilde{d}$, $C_t(\mathcal{S}) = c_s \cdot \tilde{d}^p$, and $N_{\mathcal{F}}^{\mathrm{sur}}(z_t) \triangleq \frac{\kappa(z_t)}{c_0} \cdot \ln(r_t/\epsilon_{\mathrm{KKT}})$ into the above yields

$$D(z_t) = c_s \cdot \tilde{d}^p \cdot N_{\mathcal{S}}^{\mathrm{sur}}(z_t) - c_f \cdot \tilde{d} \cdot \frac{\kappa(z_t)}{c_0} \cdot \ln\left(\frac{r_t}{\epsilon_{\mathrm{KKT}}}\right). \tag{25}$$

The PD-IPM solver $\mathcal{S}$ has a lower predicted runtime if $D(z_t) < 0$, or equivalently,

$$c_s \cdot \tilde{d}^p \cdot N_{\mathcal{S}}^{\mathrm{sur}}(z_t) < c_f \cdot \tilde{d} \cdot \frac{\kappa(z_t)}{c_0} \cdot \ln\left(\frac{r_t}{\epsilon_{\mathrm{KKT}}}\right). \tag{26}$$

For a non-convergent state, it holds $r_t > \epsilon_{\mathrm{KKT}}$, which means $\ln(r_t/\epsilon_{\mathrm{KKT}}) > 0$. Rearranging (26) yields

$$\kappa(z_t) > \frac{c_0 c_s \cdot \tilde{d}^{p-1} N_{\mathcal{S}}^{\mathrm{sur}}(z_t)}{c_f \ln(r_t/\epsilon_{\mathrm{KKT}})} = \kappa_{\mathrm{crit}}(z_t).$$

Thus, $\kappa(z_t) > \kappa_{\mathrm{crit}}(z_t)$ holds if and only if $\mathcal{S}$ has a lower predicted runtime. Conversely, $\mathcal{F}$ necessitates a smaller predicted runtime if $\kappa(z_t) < \kappa_{\mathrm{crit}}(z_t)$. This proves the desired results. □

A.3 Proof of Theorem 2

Preliminaries: Recall that $B_t \triangleq T - t + 1$. Define the certified region

$$\mathcal{C} \triangleq \{z_t \,|\, r_t \le \eta(z_t)/2,\, N_{\mathcal{F}}^{\mathrm{sur}}(z_t) \le B_t,\, N_{\mathcal{S}}^{\mathrm{sur}}(z_t) \le B_t\}.$$

Let $e(z_t) \ge 0$ bound the runtime-difference surrogate error:

$$|D(z_t) - (\tau_{\mathcal{S}}(z_t) - \tau_{\mathcal{F}}(z_t))| \le e(z_t). \tag{27}$$

The certified excess runtimes of the two static selection functions are defined as

$$\begin{aligned} G_{\mathcal{F}}(\xi) &\triangleq \sum_{t=1}^{K} [-D(z_t) - e(z_t)]_+ \cdot \mathbf{1}(z_t \in \mathcal{C}), \\ G_{\mathcal{S}}(\xi) &\triangleq \sum_{t=1}^{K} [D(z_t) - e(z_t)]_+ \cdot \mathbf{1}(z_t \in \mathcal{C}), \end{aligned} \tag{28}$$

where $[x]_+ \triangleq \max\{0, x\}$.

Proof. Under the additive runtime model, the excess runtime of the static selection function $\pi_{\mathcal{F}}$ relative to the oracle is

$$\mathcal{T}(\pi_{\mathcal{F}}) - \mathcal{T}^* = \sum_{t=1}^{K} [\tau_{\mathcal{F}}(z_t) - \tau_{\mathcal{S}}(z_t)]_+ . \tag{29}$$

From (27), we have $\tau_{\mathcal{F}}(z_t) - \tau_{\mathcal{S}}(z_t) \ge -D(z_t) - e(z_t)$. Therefore,

$$\begin{aligned} \mathcal{T}(\pi_{\mathcal{F}}) - \mathcal{T}^* &\ge \sum_{t=1}^{K} [-D(z_t) - e(z_t)]_+ \cdot \mathbf{1}(z_t \in \mathcal{C}) \\ &= G_{\mathcal{F}}(\xi). \end{aligned} \tag{30}$$

Analogously, it holds

$$\begin{aligned} \mathcal{T}(\pi_{\mathcal{S}}) - \mathcal{T}^* &= \sum_{t=1}^{K} [\tau_{\mathcal{S}}(z_t) - \tau_{\mathcal{F}}(z_t)]_+ \\ &\ge \sum_{t=1}^{K} [D(z_t) - e(z_t)]_+ \mathbf{1}(z_t \in \mathcal{C}) \\ &= G_{\mathcal{S}}(\xi). \end{aligned} \tag{31}$$

According to the definition of the instance-level solver-selection margin, we have

$$\begin{aligned} \Gamma(\xi) &= \min\{\mathcal{T}(\pi_{\mathcal{F}}), \mathcal{T}(\pi_{\mathcal{S}})\} - \mathcal{T}^* \\ &= \min\{\mathcal{T}(\pi_{\mathcal{F}}) - \mathcal{T}^*, \mathcal{T}(\pi_{\mathcal{S}}) - \mathcal{T}^*\} \\ &\ge \min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\}. \end{aligned} \tag{32}$$

If there exists a certified state satisfying $D(z_t) > e(z_t)$ (resp. $D(z_t) < -e(z_t)$), then it holds $G_{\mathcal{S}}(\xi) > 0$ (resp. $G_{\mathcal{F}}(\xi) > 0$). Hence, certified regime crossing implies

$$\Gamma(\xi) \ge \min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\} > 0. \tag{33}$$

Taking expectations over $\xi \sim \mathcal{P}$ yields

$$\Gamma_{\mathcal{P}} \ge \mathbb{E}_{\xi \sim \mathcal{P}} [\min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\}].$$

If certified regime crossing occurs with positive probability under $\xi \sim \mathcal{P}$, the right-hand side is strictly positive, and therefore $\Gamma_{\mathcal{P}} > 0$. □

A.4 Proof of Lemma 2

Proof. Under the additive runtime model, we have $\mathcal{T}(\pi) = \sum_{t=1}^{K} \tau_{\pi(s_t)}(z_t)$ and $\mathcal{T}^* = \sum_{t=1}^{K} \min_{a\in\{\mathcal{F},\mathcal{S}\}} \tau_a(z_t)$. Therefore we have

$$\mathcal{T}(\pi) - \mathcal{T}^* = \sum_{t=1}^{K} \left(\tau_{\pi(s_t)}(z_t) - \min_{a\in\{\mathcal{F},\mathcal{S}\}} \tau_a(z_t) \right). \tag{34}$$

Recall that $\Delta\tau(z_t) \triangleq \tau_{\mathcal{S}}(z_t) - \tau_{\mathcal{F}}(z_t)$. If $\pi(s_t) = \pi^*(z_t)$, the pointwise runtime regret is zero. Otherwise, the selected solver is non-optimal. If $\pi^*(z_t) = \mathcal{F}$, then $\Delta\tau(z_t) > 0$ and the regret equals $\tau_{\mathcal{S}}(z_t) - \tau_{\mathcal{F}}(z_t) = |\Delta\tau(z_t)|$. If $\pi^*(z_t) = \mathcal{S}$, then $\Delta\tau(z_t) < 0$ and the regret equals $\tau_{\mathcal{F}}(z_t) - \tau_{\mathcal{S}}(z_t) = |\Delta\tau(z_t)|$. Hence, we can deduce that

$$\tau_{\pi(s_t)}(z_t) - \min_{a\in\{\mathcal{F},\mathcal{S}\}} \tau_a(z_t) = |\Delta\tau(z_t)| \cdot \mathbf{1}\{\pi(s_t) \neq \pi^*(z_t)\}. \tag{35}$$

Summing up the above equality over $t = 1, \ldots, K$ proves (17). □

A.5 Proof of (18)

Recall that $q_t = 2y_t^{\text{rank}} - 1$, $d_{\theta,t} = \widehat{y}_\theta^{\text{time}}(s_t, \mathcal{S}) - \widehat{y}_\theta^{\text{time}}(s_t, \mathcal{F})$, and define the oracle preference label as

$$q_t^* \triangleq 2 \cdot \mathbf{1}\{\Delta\tau(z_t) > 0\} - 1. \tag{36}$$

Thus, $q_t = +1$ indicates that the supervised target favors $\mathcal{F}$, while $q_t^* = +1$ indicates that the runtime oracle favors $\mathcal{F}$. A disagreement between the learned selection $\pi_\theta(s_t)$ and the oracle selection $\pi^*(z_t)$ can occur only if either the learned score disagrees with the supervised ranking target or the supervised target disagrees with the oracle preference. Therefore we have

$$\mathbf{1}\{\pi_\theta(s_t) \neq \pi^*(z_t)\} \leq \mathbf{1}\{q_t \cdot d_{\theta,t} \leq 0\} + \mathbf{1}\{q_t \neq q_t^*\}. \tag{37}$$

Applying (37) to Lemma 2, multiplying by $|\Delta\tau(z_t)|$, summing over t, and taking expectation over $\xi \sim \mathcal{P}$ yields

$$\begin{aligned} \mathcal{E}_{\mathcal{P}}(\pi_\theta) \leq\ & \underbrace{\mathbb{E}_{\xi\sim\mathcal{P}}\left[\sum_{t=1}^{K} |\Delta\tau(z_t)| \cdot \mathbf{1}\{q_t \cdot d_{\theta,t} \leq 0\}\right]}_{\mathcal{E}_{\text{learn}}(\pi_\theta)} \\ & + \underbrace{\mathbb{E}_{\xi\sim\mathcal{P}}\left[\sum_{t=1}^{K} |\Delta\tau(z_t)| \cdot \mathbf{1}\{q_t \neq q_t^*\}\right]}_{\mathcal{E}_{\text{target}}}, \end{aligned} \tag{38}$$

which is (18). In the above $\mathcal{E}_{\text{learn}}(\pi_\theta)$ measures the runtime-weighted solver-ordering error relative to the supervised ranking target, whereas $\mathcal{E}_{\text{target}}$ measures the runtime-weighted mismatch between the supervised target and the runtime-oracle preference.

A.6 Proof of Lemma 3

Proof. The proof proceeds in three steps.

Step 1: Reduction to Average State-Distribution Error.

By the definition of $\mathcal{E}_{\text{learn}}(\pi_\theta)$, i.e., (38), we have

$$\mathcal{E}_{\text{learn}}(\pi_\theta) = \mathbb{E}_{\xi\sim\mathcal{P}}\left[\sum_{t=1}^{K} |\Delta\tau(z_t)| \cdot \mathbf{1}\{q_t \cdot d_{\theta,t} \leq 0\}\right].$$

Since $|\Delta\tau(z_t)| \leq \bar{\Delta}$, let $q(s)$ denote the supervised ranking label associated with state s, with $q(s_t) = q_t$, and let $d_\theta(s_t) = d_{\theta,t}$. Then,

$$\mathcal{E}_{\text{learn}}(\pi_\theta) \leq \bar{\Delta}K \cdot \mathbb{E}_{s\sim Z_{\pi_\theta}}\left[\mathbf{1}\{q(s) \cdot d_\theta(s) \leq 0\}\right]. \tag{39}$$

Step 2: Distribution Shift via Concentrability.

By Assumption 1, it holds $Z_{\pi_\theta}(s) \le \beta \cdot Z_{\text{data}}(s)$ for all s. Since the indicator is non-negative, we have

$$\mathbb{E}_{s\sim Z_{\pi_\theta}}\left[\mathbf{1}\{q(s)\cdot d_\theta(s)\le 0\}\right] \le \beta \cdot \mathbb{E}_{s\sim Z_{\text{data}}}\left[\mathbf{1}\{q(s)\cdot d_\theta(s)\le 0\}\right]. \tag{40}$$

Step 3: Ranking Error Bound.

With $q(s)\in\{-1,+1\}$, the binary ranking loss can be written as

$$\ell_{\text{rank}}(s;\theta) = \text{softplus}(-q(s)\cdot d_\theta(s)). \tag{41}$$

where $\text{softplus}(u) \triangleq \ln(1+e^u)$. If $q(s)\cdot d_\theta(s)\le 0$, then we have $-q(s)\cdot d_\theta(s)\ge 0$. Since $\text{softplus}(u)=\ln(1+e^u)\ge \ln 2$ for $u\ge 0$, we have

$$\mathbf{1}\{q(s)\cdot d_\theta(s)\le 0\} \le \frac{1}{\ln 2}\cdot \ell_{\text{rank}}(s;\theta). \tag{42}$$

Taking expectations over Z_{data} yields

$$\mathbb{E}_{s\sim Z_{\text{data}}}\left[\mathbf{1}\{q(s)\cdot d_\theta(s)\le 0\}\right] \le \frac{\mathcal{L}^{\text{pop}}_{\text{rank}}(\theta)}{\ln 2}.$$

Using Assumption 1, namely,

$$\mathcal{L}^{\text{pop}}_{\text{rank}}(\theta) \le \mathcal{L}^{\text{train}}_{\text{rank}}(\theta) + \epsilon_{\text{gen}},$$

we obtain

$$\mathbb{E}_{s\sim Z_{\text{data}}}\left[\mathbf{1}\{q(s)\cdot d_\theta(s)\le 0\}\right] \le \frac{\mathcal{L}^{\text{train}}_{\text{rank}}(\theta)+\epsilon_{\text{gen}}}{\ln 2}.$$

Combining (39) to (42) yields

$$\mathcal{E}_{\text{learn}}(\pi_\theta) \le \frac{\bar{\Delta}K\beta}{\ln 2}\left(\mathcal{L}^{\text{train}}_{\text{rank}}(\theta)+\epsilon_{\text{gen}}\right),$$

which is the desired result. □

A.7 Proof of Theorem 3

Proof. Define $\underline{\Gamma}_{\mathcal{P}} \triangleq \mathbb{E}_{\xi\sim\mathcal{P}}\left[\min\{G_{\mathcal{F}}(\xi), G_{\mathcal{S}}(\xi)\}\right]$. According to Theorem 2, it holds $\Gamma_{\mathcal{P}} \ge \underline{\Gamma}_{\mathcal{P}}$. Moreover, from (18) and Lemma 3, we have

$$\begin{aligned}\mathcal{E}_{\mathcal{P}}(\pi_\theta) &\le \mathcal{E}_{\text{learn}}(\pi_\theta) + \mathcal{E}_{\text{target}} \\ &\le \frac{\bar{\Delta}K\beta}{\ln 2}\left(\mathcal{L}^{\text{train}}_{\text{rank}}(\theta)+\epsilon_{\text{gen}}\right) + \mathcal{E}_{\text{target}}.\end{aligned} \tag{43}$$

Under (21), the right-hand side is strictly smaller than $\underline{\Gamma}_{\mathcal{P}}$, which implies

$$\mathcal{E}_{\mathcal{P}}(\pi_\theta) < \underline{\Gamma}_{\mathcal{P}} \le \Gamma_{\mathcal{P}}. \tag{44}$$

Based on

$$\Gamma_{\mathcal{P}} = \mathbb{E}_{\xi\sim\mathcal{P}}\left[\min\{\mathcal{T}(\pi_{\mathcal{F}}), \mathcal{T}(\pi_{\mathcal{S}})\} - \mathcal{T}^*\right], \tag{45}$$

we can obtain

$$\begin{aligned}\mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_\theta)] &= \mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}^*] + \mathcal{E}_{\mathcal{P}}(\pi_\theta) \\ &< \mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}^*] + \Gamma_{\mathcal{P}} \\ &= \mathbb{E}_{\xi\sim\mathcal{P}}\left[\min\{\mathcal{T}(\pi_{\mathcal{F}}), \mathcal{T}(\pi_{\mathcal{S}})\}\right] \\ &\le \min\left\{\mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_{\mathcal{F}})], \mathbb{E}_{\xi\sim\mathcal{P}}[\mathcal{T}(\pi_{\mathcal{S}})]\right\}.\end{aligned} \tag{46}$$

Thus, π_θ achieves a lower expected runtime than static single solvers under the stated conditions. □

B SUPPLEMENTARY EXPERIMENTAL DETAILS

This section provides additional details on the benchmark construction, numerical implementations, and evaluation protocols to complement Section 7.

B.1 AC OPTIMAL POWER FLOW

Problem Formulation: We first describe the AC optimal power flow (AC-OPF) problem and the benchmark used in our experiments. We follow the standardized formulation and benchmark library of (Babaeinejadsarookolaee et al., 2019). AC-OPF is a fundamental optimization problem in power system operation and planning. Its goal is to determine the optimal generator dispatch and voltage setpoints while satisfying the nonlinear AC power-flow equations and equipment operating limits, typically minimizing total generation cost. The problem can also be formulated with alternative objectives, such as loss minimization, emission reduction, or redispatch cost. Mathematically, AC-OPF is a nonconvex nonlinear program and is NP-hard in general.

Let $\mathcal{B}$ and $\mathcal{G}$ denote the sets of buses and generators, respectively. The optimization variables are the voltage angles θ_i, voltage magnitudes V_i, active generator outputs P_g^G, and reactive generator outputs Q_g^G. A standard formulation is

$$\min_{\theta, V, P^G, Q^G} \quad \sum_{g \in \mathcal{G}} f_g(P_g^G) \tag{47}$$

$$\text{s.t.} \quad P_i(V,\theta) = \sum_{g \in \mathcal{G}_i} P_g^G - P_{D_i},\ \forall i \in \mathcal{B}, \tag{48}$$

$$Q_i(V,\theta) = \sum_{g \in \mathcal{G}_i} Q_g^G - Q_{D_i},\ \forall i \in \mathcal{B}, \tag{49}$$

$$P_g^{\min} \le P_g^G \le P_g^{\max},\ \forall g \in \mathcal{G}, \tag{50}$$

$$Q_g^{\min} \le Q_g^G \le Q_g^{\max},\ \forall g \in \mathcal{G}, \tag{51}$$

$$V_i^{\min} \le V_i \le V_i^{\max},\ \forall g \in \mathcal{B}, \tag{52}$$

$$|S_{ij}(V,\theta)| \le S_{ij}^{\max},\ (i,j) \in \mathcal{E}, \tag{53}$$

$$\theta_{\text{ref}} = 0, \tag{54}$$

where $f_g(P_g^G) \triangleq a_g + b_g \cdot P_g^G + c_g \cdot (P_g^G)^2$ is the quadratic generation-cost function, $P_i(V,\theta)$ and $Q_i(V,\theta)$ are the AC active- and reactive-power injection functions, P_{D_i} and Q_{D_i} are the active and reactive demands, $\mathcal{E}$ is the set of branches, S_{ij} denotes the complex power flow on branch (i,j), and $\theta_{\text{ref}} = 0$ fixes the reference-bus angle to eliminate rotational symmetry. The nonconvexity of AC-OPF arises from the AC power-flow equations, which involve trigonometric functions of voltage angles and quadratic products of voltage magnitudes. The computational performance depends strongly on the local geometry of the problem. In particular, the local KKT condition number can vary substantially across different regions of the optimization landscape. A solver that is efficient in one region may make slow progress in another, which motivates adaptive solver selection along a single optimization trajectory. AC-OPF is therefore a natural benchmark for studying dynamic solver selection.

To enable reproducible and fair comparison across algorithms, the IEEE PES PGLib-OPF Task Force introduced a standardized AC-OPF formulation and an open benchmark library, PGLib-OPF (Babaeinejadsarookolaee et al., 2019). The library contains networks ranging from small systems, such as 3-bus cases, to large real-world grids with thousands of buses, on which a size-agnostic selection function can be trained and evaluated across different scales and topologies. Each network includes complete data for buses, generators, loads, branches, and transformers. The library is openly licensed, continuously maintained, and does not provide precomputed solutions. This makes it suitable both for benchmarking optimization algorithms and for generating supervised-learning targets, since users can solve the same instances with different solvers and compare runtime, convergence, and solution quality. The standardized formulation also avoids ambiguities in variable choices, constraint modeling, and data conventions that previously hindered direct comparisons across studies.

Cheap solver: Consensus ADMM (F2). The cheap solver is selected as a 2-iteration consensus ADMM procedure. The key idea of ADMM is to introduce auxiliary variables that replicate the coupling quantities, so that each subproblem depends only on a small subset of the original variables. The global coupling is then enforced through a consensus variable and dual updates.

Consider the AC-OPF problem in the following compact form:

$$\min_{x} f(x) \tag{55}$$

$$\text{s.t. } x \in \Omega, \tag{56}$$

where x collects all voltage magnitudes, voltage angles, and generator active and reactive outputs, and Ω denotes the feasible set defined by the AC power-balance equations, generator limits, voltage limits, and branch-flow limits. To apply ADMM, we partition the network into M blocks, indexed by $m \in \mathcal{M}$. Each block may correspond to a generator, a branch, a bus, or a group of components. We then introduce a local variable u_m for each block as well as a global consensus variable w. The local variable u_m is linked to w through linear coupling constraints:

$$u_m = E_m w,\ m \in \mathcal{M}, \tag{57}$$

where E_m is a selection matrix that extracts from w the entries replicated by block m. The original problem is rewritten in the following equivalent split form:

$$\min_{\{u_a m\}, w} \quad \sum_{m \in \mathcal{M}} \phi_m(u_m) + I_{\mathcal{C}}(w), \tag{58}$$

$$\text{s.t.} \quad u_m = E_m w,\ m \in \mathcal{M}, \tag{59}$$

where ϕ_m contains the local objective and local constraints associated with block m, and $I_{\mathcal{C}}$ is the indicator function of the consensus set $\mathcal{C}$. The set $\mathcal{C}$ enforces the global AC power-balance equations and the consistency of replicated voltage and power variables across blocks. The augmented Lagrangian for the split problem is given as

$$\mathcal{J}(\{u_m\}, w, \{\Lambda_m\}) \triangleq \sum_{m \in \mathcal{M}} \phi_m(u_m) + I_{\mathcal{C}}(w) + \sum_{m \in \mathcal{M}} \Lambda_m^T (u_m - E_m w) + \frac{\rho_m}{2} \|u_m - E_m w\|_2^2 \tag{60}$$

where Λ_m is the Lagrangian multiplier, and ρ_m is a penalty parameter. ADMM minimizes this augmented Lagrangian by alternating between the following three steps at each iteration:

$$u_m^{k+1} = \arg\min_{u_m} \phi_m(u_m) + (\Lambda_m^k)^T \cdot (u_m - E_m w^k) + \frac{\rho_m}{2} \|u_m - E_m w^k\|_2^2,\ m \in \mathcal{M}, \tag{61}$$

$$w^{k+1} = \arg\min_{w} I_{\mathcal{C}}(w) + \sum_{m \in \mathcal{M}} \Lambda_m^T (u_m^{k+1} - E_m w) + \frac{\rho_m}{2} \|u_m^{k+1} - E_m w\|_2^2 \tag{62}$$

$$\Lambda_m^{k+1} = \Lambda_m^k + \rho_m \cdot (u_m^{k+1} - E_m w^{k+1}),\ m \in \mathcal{M}, \tag{63}$$

The structure of the local subproblems depends on the block type.

- **Generator Block**: For a generator block, the local variables are the active and reactive power outputs P_g^G and Q_g^G. The local objective is the quadratic generation cost $f_g(P_g^G)$, and the local constraints are the generator limits. The subproblem is a small quadratic program with box constraints, which admits a closed-form proximal solution followed by projection onto the limits. Specifically, the active-power update is the proximal map of the quadratic cost, and the reactive-power update is a simple projection onto $[Q_g^{\min}, Q_g^{\max}]$.
- **Branch Blocks**: For a branch block, the local variables are the voltage magnitudes and angles at the two endpoint buses. The local constraints include the nonlinear AC branch-flow equations and the branch-flow limits. Because this subproblem is nonconvex, it is not solved to global optimality. Instead, a single damped Gauss-Newton correction is applied to the endpoint voltages, and the branch-flow quantities are then evaluated from the updated voltages. This makes the branch update the only inexact part of the local minimization, and it keeps the per-iteration cost low.
- **Consensus Update** The consensus variable w typically represents the bus voltage magnitudes, voltage angles, and nodal power injections. The set $\mathcal{C}$ enforces the AC power-balance equations at each bus and the agreement among replicated variables from different blocks. Because $\mathcal{C}$ is affine, the w-update is a quadratic projection that decomposes into independent small systems per bus, each of which can be solved efficiently.

Heavy Solver: Primal-dual interior-point method (S1)**.** The Heavy action is chosen as a one-iteration primal-dual interior-point method (PD-IPM) applied directly to the nonlinear AC-OPF problem. PD-IPM treats the full AC-OPF problem as a single nonlinear program and solves a sequence of perturbed KKT systems by Newton's method. Each iteration is computationally expensive because it requires assembling the Lagrangian Hessian and constraint Jacobians and solving a large sparse linear system.

To implement PD-IPM, we write the AC-OPF problem in the following generic nonlinear programming form

$$\min_x f(x) \tag{64}$$

$$\text{s.t. } h(\boldsymbol{x}) = 0, \tag{65}$$

$$g(x) \leq 0, \tag{66}$$

where the optimization variable is

$$x \triangleq [\theta\ V\ P^G\ Q^G], \tag{67}$$

θ and V collect the voltage angles and magnitudes, and P^G and Q^G collect the active and reactive generator outputs. The equality constraints $h(x) = 0$ include the AC active- and reactive-power balance equations at all buses, together with the reference-bus angle constraint. The inequality constraint $g(x) \leq 0$ include generator active- and reactive-power limits, bus voltage-magnitude limits, branch-flow limits, and branch angle-difference limits. The objective $f(x)$ is the total generation cost. PD-IPM introduces a slack variable $s > 0$ for the inequality constraints and rewrites $g(x) < 0$ as

$$g(x) + s = 0,\ s > 0. \tag{68}$$

Let y denote the Lagrangian multiplier for the equality constraint $h(x) = 0$, and also let $z > 0$ denote the multiplier for the inequality constraints. The perturbed KKT conditions are

$$\nabla f(x) + J_h(x)^T \cdot y + J_g(x)^T \cdot z = 0, \tag{69}$$

$$h(x) = 0 \tag{70}$$

$$g(x) + s = 0 \tag{71}$$

$$S \cdot z = \mu \cdot e \tag{72}$$

where $J_h(x)$ (resp. $J_g(x)$) is the Jacobian of $h(x)$ (resp. $g(x)$), $S = \text{diag}\{s\}$, e is the vector of all ones, and μ is the barrier parameter. The last equation is the centering condition. As $\mu \to 0$, the solution of the perturbed KKT system approaches a KKT point of the original AC-OPF problem.

At each PD-IPM iteration, we linearize the perturbed KKT conditions around the current iterate (x, y, z, s) and solve for the Newton direction $(\Delta x, \Delta y, \Delta z, \Delta s)$. Define the residuals as

$$r_d = \nabla f(x) + J_h(x)^T \cdot y + J_g(x)^T \cdot z \tag{73}$$

$$r_p = h(x) \tag{74}$$

$$r_i = g(x) + s \tag{75}$$

$$r_c = S \cdot z - \mu \cdot e \tag{76}$$

The Newton system is

$$\begin{bmatrix} \text{Hess}(x) & J_h(x)^T & J_g(x)^T & 0 \\ J_h(x) & 0 & 0 & 0 \\ J_g(x) & 0 & 0 & 0 \\ 0 & 0 & S & Z \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta s \end{bmatrix} \tag{77}$$

where $Z = \text{diag}\{\text{z}\}$, $\boldsymbol{I}$ is the identity matrix, and $\text{Hess}(x)$ is the regularized Hessian of the Lagrangian. Solving this linear equation yields $(\Delta x, \Delta y, \Delta z, \Delta s)$. The variables are then updated as

$$x^{k+1} = x^k + \alpha_p \cdot \Delta x, \tag{78}$$

$$s^{k+1} = s^k + \alpha_s \cdot \Delta s, \tag{79}$$

$$y^{k+1} = y^k + \alpha_d \cdot \Delta y, \tag{80}$$

$$z^{k+1} = z^k + \alpha_d \cdot \Delta z. \tag{81}$$

The barrier parameter is updated as $\mu \leftarrow \sigma \cdot \mu$, where $\sigma \in (0, 1)$ is the centering parameter. Alternatively, a Mehrotra predictor-corrector variant (Tapia et al., 1996) can also be used to adapt the barrier parameter automatically.

Solver Switching on a Shared Optimization Trajectory. The Cheap and Heavy solver are not independent solvers that restart the AC-OPF problem from scratch. Instead, they operate on the same evolving optimization trajectory, and switching between them continues the current iterate rather than reinitializing the problem. This design is essential for the selection: it allows the selection function to exploit the complementary strengths of ADMM and PD-IPM without discarding the progress made by either method.

B.2 Quadratic Programming

Problem Formulation. We consider convex quadratic programming (QP) problems of the form

$$\begin{aligned} \min_x \quad & \frac{1}{2} x^T P x + q^T x \\ \text{s.t.} \quad & Ax = b, \\ & Gx \le h, \\ & \ell \le x \le u, \end{aligned} \tag{82}$$

where P is a symmetric and positive semidefinite matrix, A and G are constraint matrices. The vectors q, b, h, l, and u collect the linear objective coefficients, right-hand sides, and variable bounds, respectively. Since P is positive semidefinite, the problem is convex and its global optimality is completely characterized by the KKT conditions.

Benchmark Instances. We construct the QP benchmark from the Maros–Mészáros benchmark (Maros & Mészáros, 1999). The repository contains 138 convex QP problems collected from multiple sources, together with a standard QPS format that extends the classical MPS format with a quadratic objective section. The repository was introduced to provide a common reference set for evaluating and comparing QP solvers, in the same spirit as the NETLIB LP test set for linear programming. Each problem is distributed with its constraint matrices, objective data, variable bounds, and sparsity structure.

Cheap Solver: Over-relaxed ADMM (F1). The Cheap solver is selected as a 1-iteration proximal, over-relaxed ADMM algorithm (Boyd et al., 2011). Let $\mathcal{I}_b$ denote the indices of variables having at least one finite bound, and let S_b be the corresponding selector matrix. Define

$$K = \begin{bmatrix} A \\ G \\ S_b \end{bmatrix}, \qquad \mathcal{Z} = \{b\} \times (-\infty, h] \times [\ell_{\mathcal{I}_b}, u_{\mathcal{I}_b}], \tag{83}$$

where either endpoint of the bound interval may be infinite. Variables with neither a finite lower nor a finite upper bound are omitted from the bound block. The QP can then be written in the following form:

$$\min_{x,w} \frac{1}{2} x^T P x + q^T x + \iota_{\mathcal{Z}}(w), \tag{84}$$

$$\text{s.t. } Kx - w = 0, \tag{85}$$

where $\iota_{\mathcal{Z}}$ is the indicator function corresponding to $\mathcal{Z}$. The augmented Lagrangian function of the above problem is given as

$$\mathcal{J}(x, w, \Lambda) = \frac{1}{2} x^T P x + q^T x + \iota_{\mathcal{Z}}(w) + \langle \Lambda, Kx - w \rangle + \frac{\sigma}{2} \|Kx - w\|_2^2 \tag{86}$$

based on which we obtain the over-relaxed ADMM algorithm:

$$\tilde{x}^{k+1} = \arg\min_x \ \frac{1}{2} x^T P x + q^T x + \langle \Lambda^k, Kx - w^k \rangle + \frac{\rho}{2} \|Kx - w^k\|_2^2 + \frac{\sigma}{2} \|x - x^k\|_2^2, \tag{87}$$

$$x^{k+1} = \alpha \cdot \tilde{x}^{k+1} + (1-\alpha) \cdot x^k, \tag{88}$$

$$w^{k+1} = \arg\min_{w} \ \iota_{\mathcal{Z}}(w) + \langle \Lambda^k, K\tilde{x}^{k+1} - w \rangle + \frac{\rho}{2}\|K\tilde{x}^{k+1} - w\|_2^2, \tag{89}$$

$$\Lambda^{k+1} = \Lambda^k + \rho(Kx^{k+1} - w^{k+1}), \tag{90}$$

The solution to the $\tilde{x}^{k+1}$-subproblem can be readily obtained by taking the gradient of the objective function and set it to 0, namely,

$$\tilde{x}^{k+1} = (P + \sigma I + \rho K^T K)^{-1} \cdot \Big(\sigma \cdot x^k - q + K^T\big(\rho \cdot w^k - \Lambda^k\big)\Big) \tag{91}$$

The solution to the $\tilde{w}^{k+1}$-subproblem is given as

$$w^{k+1} = \text{Proj}_{\mathcal{Z}}\Big\{K\tilde{x}^{k+1} + \frac{\Lambda^k}{\rho}\Big\} \tag{92}$$

The hyperparameters are configured as

$$\rho = 0.03, \qquad \alpha = 1.4, \qquad \sigma = 10^{-8}. \tag{93}$$

For a fixed QP instance, the coefficient matrix

$$P + \sigma I + \rho K^\top K \tag{94}$$

is constant and is therefore assembled and factorized only once. The resulting factorization is reused by all subsequent ADMM actions along that instance trajectory.

Heavy Solver: Mehrotra primal-dual interior-point method (S1)**.** To implement PD-IPM, first note that all inequalities and finite variable bounds can be assembled as

$$Cx \le d. \tag{95}$$

Introducing a slack variables $s > 0$, a Lagrangian multiplier $z > 0$ associated to the inequality constraint, and a Lagrangian multiplier y associated to the equality constraint, then the KKT residual equation can be written as

$$r_{\mathrm{d}} = Px + q + A^\top y + C^\top z, \tag{96}$$

$$r_{\mathrm{p}} = Ax - b, \tag{97}$$

$$r_{\mathrm{i}} = Cx + s - d, \tag{98}$$

$$r_{\mathrm{c}} = Sz, \tag{99}$$

where $S = \mathrm{diag}(s)$. The Heavy solver is chosen as an Mehrotra predictor-corrector PD-IPM method. The implement details are similar as those in Appendix B.1

B.3 Solver Selector Training Details

The QP and ACOPF implementations follow the state representation, prediction targets, training objective, and dataset-aggregation procedure described in Section 4. We provide the implementation details used to train the selector checkpoints reported in the main experiments.

Network architecture and optimization. For both QP and ACOPF, our neural network is implemented as a multilayer perceptron with widths $8 \to 128 \to 64 \to 32 \to 4$. SiLU activations are applied after each of the three hidden layers. The network contains $11{,}620$ trainable parameters, and its four outputs correspond to the reachability and remaining-runtime predictions for the Cheap and Heavy actions. Input features are standardized using statistics computed from the training split.

We train the solver selector using AdamW with learning rate 10^{-3}, weight decay 10^{-5}, and gradient-norm clipping at 5. The tradeoff parameters in Eq. (7) are set as

$$(\lambda_{\mathrm{reach}}, \lambda_{\mathrm{time}}, \lambda_{\mathrm{rank}}) = (2, 1, 1)$$

for both domains. The threshold in the Huber loss in Eq. (9) is set to 0.5. The failure-penalty parameter is $\lambda_{\mathrm{fail}} = 384$ for QP and $\lambda_{\mathrm{fail}} = 720$ for ACOPF. For QP, we use a batch size of $4{,}096$ and train for 250 epochs. For ACOPF, we use a batch size of $32{,}768$ and train for 500 epochs.

Dataset aggregation. For QP, the initial training set contains $129{,}115$ paired states, followed by one dataset aggregation round that contributes $18{,}489$ additional on-policy states, giving $147{,}604$ paired states in total. The aggregation round uses 200 training instances from each of the five QP structures, with three trajectories per instance and up to eight sampled states per trajectory.

For ACOPF, the initial training set contains $38{,}215$ paired states. Three dataset aggregation rounds contribute $2{,}598$, $7{,}465$, and $1{,}611$ additional states, respectively, giving $49{,}889$ paired states in total. The first two aggregation rounds use 200 training instances per topology, with three trajectories per instance and at most eight sampled states per trajectory.

During dataset aggregation, QP uses exploration probabilities $\epsilon \in \{0, 0.2, 0.6\}$ across the collected trajectories. For ACOPF, the exploration probabilities are $\{0.2, 0.6, 1.0\}$ in the first collection stage and $\{0, 0.2, 0.6\}$ thereafter. When exploration is triggered, the Heavy action is sampled with probability 0.8.

B.4 Reference Solutions and Solver-Selector Label Generation

We construct high-accuracy reference solutions offline for evaluating convergence and solution quality. These references are used only for dataset construction and terminal evaluation, and are not provided to the solver selector.

For QP, we construct five parametric problem families from the Maros–Mészáros benchmark (Maros & Mészáros, 1999). Within each family, the sparse problem structure and dimensions are fixed, while the linear objective and constraint right-hand sides are jointly perturbed. The perturbation is constructed around a feasible witness so that every generated instance remains feasible. Each generated QP is independently solved to high accuracy using Clarabel (Goulart & Chen, 2026) and PIQP (Schwan et al., 2023). We retain instances for which the two solvers return consistent high-accuracy solutions, and use the solution with the smaller independently evaluated KKT residual as the reference. The same frozen reference is used when evaluating all methods.

For ACOPF, operating points are generated from the PGLib cases using the PGLearn data-generation pipeline (Klamkin et al., 2025). Loads are perturbed using PGLearn's standard sampling procedure while preserving the original network topology and operating limits. Each generated instance is solved offline with Ipopt (Wächter & Biegler, 2006) using a high-accuracy configuration. Since ACOPF is nonconvex, the resulting solution is treated as a high-accuracy local reference rather than a certified global optimum. Only instances whose returned solutions satisfy the prescribed feasibility and KKT checks are retained.

C Additional Experimental Results

C.1 Visualization of Experimental Results

Fig. 2 and Fig. 3 provide a visual summary of the main experimental results in Section 7. They compare the strict convergence rate and runtime of different methods on the QP and ACOPF benchmarks, respectively.

C.2 Generalization to Unseen Problem Scales and Structures

We evaluate whether the solver selector equipped with the fixed-dimensional state representation generalizes beyond the problem scales and structures used for training. All transfer evaluations reported in this subsection are zero-shot: the solver selectors' parameters are frozen before evaluation on the target cases, and no target-specific fine-tuning, on-policy data aggregation, action prefix, threshold, fallback rule, or feature transformation is introduced. The network architecture and input representation remain unchanged across problem sizes. Each evaluation starts from a fresh numerical solver state and uses CPU-based model inference. Unless otherwise stated, speedups are computed only on instances successfully reached by both Heavy solver and our proposed solver selector. ACOPF uses the 10^{-4} endpoint criterion, whereas QP uses the 10^{-6} criterion. Throughout this paper, size-agnostic refers to the fact that our method's input dimension and network architecture are independent of the number of optimization variables, constraints, buses, or matrix dimensions. It does not imply uniform performance over arbitrary problem sizes; rather, we evaluate whether

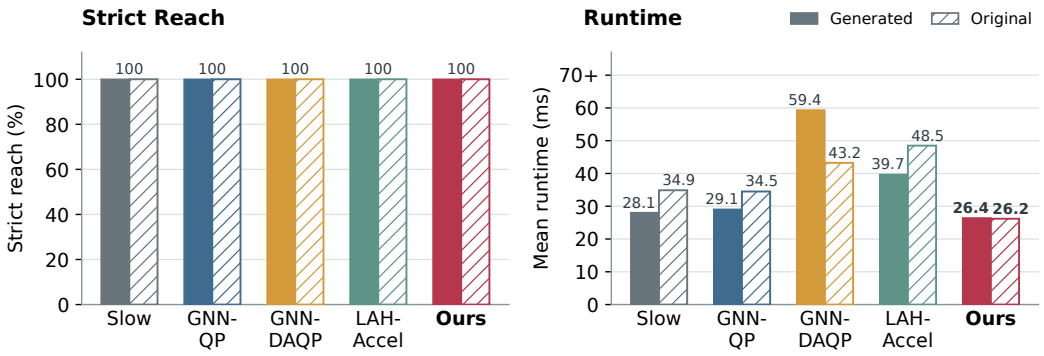


Figure 2: QP strict reach and runtime.

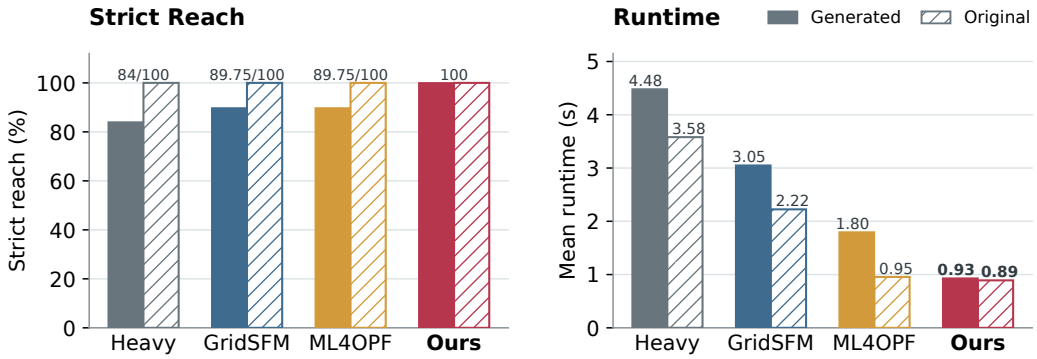


Figure 3: ACOPF strict reach and runtime.

the same frozen solver selector can be deployed zero-shot on unseen problem scales and structures without architectural modification or target-specific retraining.

ACOPF: unseen topology and scale. We first perform leave-one-topology-out evaluation over the four main PGLib topologies. In each fold, the target topology—and therefore its corresponding network scale—is excluded from both supervised fitting and on-policy data aggregation. The resulting solver selector is then frozen and evaluated directly on the held-out topology without any adaptation. This setting tests zero-shot transfer simultaneously across topology and problem scale.

We additionally freeze the proposed method's network trained on all four main topologies and deploy the same model, without refitting or target-specific model selection, on four additional PGLib cases whose topologies and network sizes are absent from solver selector training.

Our solver selector preserves 100% reach in all four leave-one-topology-out folds, even though the target topology and its associated problem scale are absent from model training and data aggregation. On the held-out `case300_ieee` and `case500_goc` topologies, the solver selector both increases reach and achieves shared-reach speedups of $3.126\times$ and $4.706\times$, respectively. Transfer is approximately runtime-neutral on `case793_goc`, while the solver selector is slower on `case588_sdet`. Thus, successful transfer does not require the target topology or its scale to appear during training, although the magnitude of the runtime advantage remains problem dependent.

On the four additional unseen PGLib cases, the same frozen four-topology Our proposed method preserves 100% reach and is faster than Heavy on every case, with speedups ranging from $1.028\times$ to $1.565\times$. These target networks have both topologies and sizes absent from training. Together, the two evaluations provide direct evidence of zero-shot cross-scale and cross-topology generalization without topology-specific retraining.

QP: unseen instances and matrix structures. For QP, a single solver selector is trained jointly on the generated training families and is first evaluated on held-out generated test instances. These

Table 4: ACOPF generalization to unseen topologies and problem scales. The left block reports leave-one-topology-out evaluation, where the target topology is excluded from both supervised fitting and on-policy data aggregation. The right block evaluates the same frozen solver selector trained on the four main topologies on additional unseen PGLib cases. Wall-time speedups are computed on the shared-reach subset and therefore need not equal the ratio of the two displayed mean runtimes when the two methods have different reach rates.

Leave-One-Topology-Out						**Additional Unseen Cases**					
	Reach (%)		Runtime (s)		Speedup		Reach (%)		Runtime (s)		Speedup
Case	Heavy	Ours	Heavy	Ours	Heavy/Ours	Case	Heavy	Ours	Heavy	Ours	Heavy/Ours
case300_ieee	47	**100**	3.429	**0.520**	**3.126**×	case89_pegase	**100**	**100**	0.734	**0.469**	**1.565**×
case500_goc	89	**100**	3.586	**0.666**	**4.706**×	case60_c	**100**	**100**	0.586	**0.427**	**1.373**×
case588_sdet	**100**	**100**	**1.583**	2.575	0.615×	case5_pjm	**100**	**100**	0.207	**0.194**	**1.068**×
case793_goc	**100**	**100**	**1.224**	1.229	0.995×	case197_snem	**100**	**100**	0.482	**0.469**	**1.028**×

Table 5: QP generalization on held-out generated instances and unseen original Maros-Mészáros problem structures. The same frozen solver selector is used for the unseen original cases without target-specific retraining or model selection.

Held-Out Generated Test Instances

	Reach (%)		Runtime (ms)		Speedup
Case	Heavy	Ours	Heavy	Ours	Heavy/Ours
PRIMAL1	**100**	**100**	13.260	**12.100**	**1.096**×
PRIMAL2	**100**	**100**	21.140	**19.300**	**1.095**×
PRIMAL3	**100**	**100**	56.640	**54.330**	**1.043**×
PRIMALC8	**100**	**100**	33.250	**30.050**	**1.106**×

Unseen Original QP Structures

	Reach (%)		Runtime (ms)		Speedup
Case	Heavy	Ours	Heavy	Ours	Heavy/Ours
DUAL2	**100**	**100**	11.482	**9.702**	**1.183**×
PRIMALC1	**100**	**100**	14.029	**12.443**	**1.127**×
QSCSD1	**100**	**100**	**23.420**	24.169	0.969×
VALUES	**100**	**100**	**12.616**	13.020	0.969×

test instances preserve their source matrix structures but contain numerical instances that are not used for solver selector network fitting.

We then freeze the same solver selector and deploy it directly on original Maros-Mészáros QPs that are absent from both model fitting and model selection. In this second setting, the target problems differ from the training families not only in numerical coefficients but also in their underlying matrix structures, providing an additional zero-shot test beyond the structures represented during training.

The proposed method retains 100% reach on every displayed QP. On the held-out generated test groups, it improves over Heavy solver in all four cases, with speedups ranging from $1.043\times$ to $1.106\times$. Because these instances belong to matrix families represented during training, this part primarily evaluates generalization to unseen numerical instances under a single shared solver selector.

The unseen original Maros-Mészáros cases provide a stronger structural transfer test. Without retraining, our method improves runtime by $1.183\times$ on DUAL2 and $1.127\times$ on PRIMALC1, while QSCSD1 and VALUES remain within 3.1% of Heavy. Full reach is preserved in all four cases. Hence, zero-shot deployment is not restricted to the generated matrix structures seen during network training, although the realized runtime gain depends on the numerical regime of the target problem.

Overall, these experiments complement the multi-scale results in the main text in two ways. First, leave-one-topology-out ACOPF evaluation shows that the selector can transfer to a problem scale absent from both supervised training and on-policy data aggregation. Second, direct deployment on additional PGLib networks and original Maros-Mészáros QPs shows that the same fixed-dimensional representation can be used without architectural modification on previously unseen topologies, sizes, and matrix structures. The results therefore support the intended size-agnostic design of the selector and demonstrate zero-shot generalization beyond the scales and structures represented during training.

C.3 Robustness to Alternative Cheap–Heavy Configurations

We examine whether the proposed framework depends on one particular pair of numerical actions. For each alternative configuration, we regenerate the counterfactual action labels and train a sepa-

Table 6: QP robustness under alternative Cheap–Heavy configurations.

Case	End-to-end runtime (ms)				Heavy/Ours	
	GNN-QP	Heavy	Ours B	Ours C	B	C
PRIMAL1	13.32	12.53	12.13	**10.83**	1.027×	1.164×
PRIMAL2	22.04	14.47	13.17	**12.66**	1.089×	1.151×
PRIMAL3	57.21	30.02	28.18	**26.36**	1.065×	1.138×
PRIMALC5	16.11	12.36	12.42	**11.89**	0.990×	1.045×
PRIMALC8	36.65	19.39	16.26	**16.13**	1.191×	1.203×
Average	29.07	17.75	16.43	**15.57**	1.077×	**1.143**×

Table 7: ACOPF robustness under alternative Cheap–Heavy configurations.

Case	Strict reach (%)				End-to-end runtime (s)			
	ML4OPF	Heavy	Ours B	Ours C	ML4OPF	Heavy	Ours B	Ours C
case300_ieee	65	86	**100**	**100**	2.962	1.497	**0.388**	0.390
case500_goc	95	100	**100**	**100**	1.867	1.078	**0.516**	0.522
case588_sdet	99	100	**100**	**100**	1.332	**1.104**	1.110	2.098
case793_goc	100	100	**100**	**100**	1.031	0.883	**0.825**	0.828
Average	89.75	96.50	**100**	**100**	1.798	1.140	**0.710**	0.959

rate size-agnostic solver selector from scratch using the same eight-dimensional state representation and training objective as in the main experiments. Thus, this study evaluates the robustness of the proposed framework after configuration-specific training, rather than assuming zero-shot transfer of a fixed solver selector across different solver definitions.

For QP, configurations B and C use the same relaxed-ADMM Cheap update and warm-started Mehrotra predictor–corrector Heavy update. Configuration B groups one ADMM round into each Cheap action, whereas configuration C groups three consecutive ADMM rounds into each Cheap action. Both use $\rho = 0.03$, over-relaxation $\alpha = 1.6$, primal regularization 10^{-8}, and Heavy solver with fraction-to-boundary parameter $\tau = 0.995$. A fresh numerical kernel and factorization are constructed for every QP instance, and no numerical state is reused across instances.

GNN-QP, Heavy solver, and both solver selector reach all 500 generated test QPs under the 10^{-6} endpoint criterion. Our method with configuration B is $1.077\times$ faster than its matched Heavy control and $1.769\times$ faster than GNN-QP in average runtime. Our method with configuration C is $1.143\times$ faster than its matched Heavy control and $1.867\times$ faster than GNN-QP. Because the Heavy-only numerical algorithm is identical for configurations B and C, the Heavy column reports the mean of the two independently measured Heavy-only runtime; the reported speedups with congfigurations B and C are computed using the corresponding configuration-matched Heavy solver's timing. These results show that the solver selector advantage is retained when the granularity of the Cheap action is changed from one to three ADMM rounds.

For ACOPF, both configurations retain the same split-ADMM structure and warm-started PIPS Heavy continuation. Configuration B uses a diagonally preconditioned projected-gradient correction in each branch-local update, whereas configuration C uses a damped Gauss–Newton correction. Both use $(\rho_{pq}, \rho_{v\theta}) = (50, 1.6 \times 10^5)$. To keep the held-out generated test split untouched for the primary evaluation, this auxiliary robustness study is reported on the 400 generated validation operating points.

Both ACOPF solver selector reach all 400 operating points, whereas their matched Heavy controls reach 386. On the subsets of instances reached by both the solver selector and its corresponding Heavy control, our method with configuration B and our method with configuration C are respectively $1.454\times$ and $1.081\times$ faster than Heavy solver. Across all 400 operating points, their average runtimes are also $2.534\times$ and $1.874\times$ lower than that of ML4OPF, while attaining higher strict reach. Thus, at the aggregate level, the higher-reach and lower-runtime behavior persists under alternative Cheap–Heavy realizations, including changes in Cheap-action granularity for QP and in the branch-local nonlinear correction for ACOPF.