AHEAD: ADAPTIVE NEURAL ACCELERATION FOR LARGE-SCALE LINEAR PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale linear programs with millions of variables and constraints strain time and memory budgets. First-order methods such as the primal-dual hybrid gradient scale to this regime because they avoid matrix factorizations, but they require many iterations. Learned accelerators can reduce these iterations, yet fewer iterations do not guarantee a shorter solve. Learning inference also takes time, and running the accelerator longer trades solver time for inference time. Existing unrolled accelerators handle this trade-off poorly: each layer processes every variable and constraint, the predicted iterates still need solver refinement, and every instance runs the same layers. We therefore ask: What neural accelerator design achieves end-to-end speedups on large-scale linear programs across sizes and target accuracies? We introduce AHEAD, an accelerator that predicts step sizes for first-order linear programming solvers while keeping their update rules. Its weight-shared predictor reads compact solver statistics and rescales the primal and dual step sizes, so each warm-start iteration stays inexpensive and a short warm-start phase hands the solver a warm start. A learned handoff policy sets the number of warm-start iterations for each instance and target accuracy, after which the solver finishes under its own termination criterion. On packing linear programs with up to 10^7 variables, AHEAD achieves $6.5 - 11.6\times$ end-to-end speedups over Gurobi's first-order solver from a cold start, including on instances of unseen sizes and distributions. The gains extend to PageRank linear programs and hold across difficulty levels and target accuracies. Our code is available at `https://anonymous.4open.science/r/AHEAD-43D2/`.

1 INTRODUCTION

Linear programming (LP) underpins large-scale decision-making, but instances with millions of variables and constraints strain time and memory budgets (Dantzig, 1963; Nesterov, 2014). First-order methods (FOMs) are attractive at this scale because they rely on sparse matrix-vector multiplications rather than costly factorizations (Lin et al., 2021; Applegate et al., 2023; Lu & Yang, 2025). However, they often need many iterations, particularly at high accuracy (Applegate et al., 2023). Their progress depends strongly on primal and dual step sizes: conservative choices slow convergence, while aggressive ones can destabilize it (Goldstein et al., 2013; Malitsky & Pock, 2018).

Learning-to-optimize (L2O) offers a promising way to reduce the computational cost of iterative optimization. Early learned optimizers replace hand-designed update rules with learned dynamics (Li & Malik, 2016; Andrychowicz et al., 2016; Wichrowska et al., 2017). More recent solver-integrated methods predict one-shot warm starts (Sambharya et al., 2024), unroll first-order or interior-point trajectories (Li et al., 2024; Qian et al., 2024), or learn feedback policies that tune algorithmic parameters online (Oshin et al., 2026). Across these approaches, learning exploits recurring structure among related instances to reduce the cost subsequently performed by an optimization solver.

Fewer solver iterations, however, do not guarantee a shorter solve. In learned acceleration of LP solvers, a solve typically has two stages: a neural accelerator first produces a warm start, and the classical solver then continues from it until the solution reaches the target accuracy. We call the two stages together, the neural warm start followed by the classical solve, *end-to-end neural acceleration*. Existing unrolled accelerators typically rely on a fixed number of computational layers. Each learned layer processes every variable and constraint and predicts new primal–dual iterates in place

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

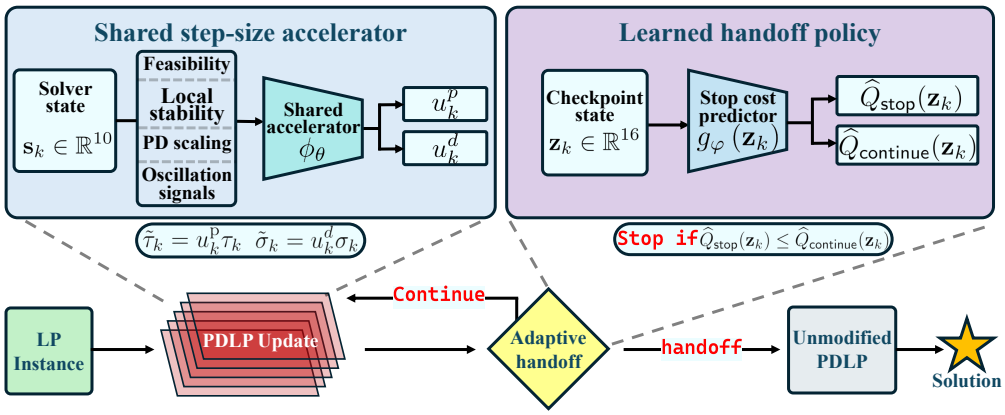


Figure 1: Framework of AHEAD.

of the solver's native update, causing inference cost to grow with both instance size and rollout depth (Qian et al., 2024; Li et al., 2024). These approximate outputs may still require additional solver running time to recover feasibility or attain high accuracy (Sambharya et al., 2024; Gao et al., 2024). Moreover, fixed-depth models use the same compute budget for every instance. This prevents them from adapting computational efforts to instance-specific needs and limits generalization across problem distributions and difficulty levels (Chen et al., 2022; Sucker & Ochs, 2025). Consequently, neural inference and subsequent refinement can offset—or even outweigh—the time saved by reducing solver iterations. These limitations lead to our central question: **What neural accelerator design achieves end-to-end speedups on large-scale LPs across sizes and target accuracies?**

To address this question, we introduce AHEAD (**A**daptive **H**orizon for **E**nd-to-end **A**cceleration of Primal–**D**ual Solvers), a lightweight neural accelerator for first-order LP solvers. In the first stage, which we call the warm-start phase, AHEAD applies a weight-shared step-size predictor at every iteration and accelerates the solver only by adjusting its step sizes. To keep each warm-start iteration cheap, the predictor reads only ten statistics built from the solver's own intermediate results and outputs one multiplier for each of the primal and dual step sizes. AHEAD's solution quickly approaches the target accuracy on its own. Because only the step sizes change, AHEAD is also fully aligned with the solver: when high accuracy is required, its solution can be handed directly to the solver as a warm start. A learned handoff policy picks this moment by predicting whether stopping now or continuing gives a shorter end-to-end time. Together, these designs address three core challenges: they let the solver reach the target accuracy in fewer iterations, keep the time spent on each iteration low, and adapt the number of warm-start iterations to each instance and accuracy.

Our main contributions are summarized as follows:

- **End-to-end speedups from fast convergence.** AHEAD's solution quickly approaches the target accuracy within a short warm-start phase, leaving little remaining solver time, which yields many-fold end-to-end speedups across unseen scales, LP families, and accuracy levels.
- **Low time per iteration.** We keep the inference time of each warm-start iteration small at any problem size: the predictor reads a 10-dimensional solver-native state and outputs two step-size multipliers, which suffice to steer PDHG through its oscillatory early phase.
- **Adaptive number of warm-start iterations.** We apply a weight-shared predictor at each iteration recurrently, so the warm-start phase can run for any number of iterations, and a learned handoff policy selects this number for each instance and accuracy, further reducing the end-to-end time.

2 Related Work

First-order methods for large-scale linear programming. Chambolle & Pock (2011) introduced PDHG, whose matrix–vector updates suit large-scale saddle-point problems. Applegate et al. (2021) developed PDLP, preconditioning, adaptive step sizes, and primal–dual balancing; subsequent work

used adaptive restarting to improve high-accuracy convergence (Applegate et al., 2023). These solvers adapt through hand-designed rules, but do not learn from related LP solves.

Learning components of LP solvers. Li et al. (2022) learn variable reorderings, while Kuang et al. (2025) learn presolve routines, including when to stop presolving. Iwata & Sakaue (2025) generate instance-specific projections to reduce LP dimension. Other works learns simplex bases or pivot decisions (Fan et al., 2023; Liu et al., 2024; Lu & Yan, 2025). These methods can accelerate stages such as presolve or simplex search, but their neural inference overhead may outweigh the solver time saved.

Learned optimizers and warm starts. Learned update rules can transfer across optimization tasks (Andrychowicz et al., 2016; Wichrowska et al., 2017), although generalization outside the training distribution remains challenging (Liu et al., 2023). Sambharya et al. (2024) learn warm starts for quadratic optimization. For LPs, Li et al. (2024) use a fixed-depth PDHG-Net to predict primal–dual states, then refine them with PDLP; they also demonstrate transfer to larger instances of related problem families. Oshin et al. (2026) learn dimension-agnostic parameter feedback for QP and SQP solvers.

Adaptive computation. Graves (2016); Banino et al. (2021) study input-dependent neural computation, while Chen et al. (2020) learn when to stop the neural network. For LPs, however, stopping decisions should exploit LP-specific feasibility and gap signals and account for the remaining solver work after learned computation ends. Taken together, prior work leaves open how to combine inexpensive control of LP iterations with instance-dependent computation to obtain consistent end-to-end gains across problem sizes and families.

3 AHEAD: Lightweight Warm Starts with Adaptive Depth

For an LP instance i, consider a learned accelerator parameterized by θ that controls the solver during a warm-start phase of K iterations, and then hands the solver a primal–dual warm start $\mathbf{z}_{i,K} = (\hat{\mathbf{x}}_i^K, \hat{\mathbf{y}}_i^K)$; we call this step the *handoff*. The solver continues from $\mathbf{z}_{i,K}$ until it reaches the target accuracy ϵ. We define the end-to-end time of instance i is

$$\mathcal{T}_i(\theta, K; \epsilon) = T_{\text{warm}}(K) + T_{\text{solver}}(\mathbf{z}_{i,K}; \epsilon), \tag{1}$$

where the warm-start time $T_{\text{warm}}(K)$ covers the K warm-start iterations, including the inference of the predictor and the handoff policy, and the remaining solver time $T_{\text{solver}}(\mathbf{z}_{i,K}; \epsilon)$ covers the solver's run from the warm start until it reaches ϵ. The cold-start time $\mathcal{T}_{i,\text{cold}} = \mathcal{T}_i(\theta, 0; \epsilon)$ covers the same run from the solver's default initial point.

AHEAD yields an end-to-end speedup on instance i if $\mathcal{T}_i(\theta, K_i; \epsilon) < \mathcal{T}_{i,\text{cold}}$, where K_i is the number of warm-start iterations selected for instance i. Our goal is to achieve this on all or most instances, so we design AHEAD to keep both terms of Equation (1) small. Section 3.1 derives what each term requires of the accelerator. Based on this analysis, AHEAD consists of two learned components: a weight-shared step-size predictor f_θ (Section 3.2) that adjusts the primal and dual step sizes during the warm-start phase, and a learned handoff policy g_φ (Section 3.3) that selects the number of warm-start iterations K_i for each instance and accuracy. Section 3.4 describes how we train the predictor over varying numbers of warm-start iterations, and Figure 1 summarizes the framework.

3.1 Design Requirements

Reducing $\mathcal{T}_i$ requires keeping both terms of Equation (1) small. For the warm-start phase to pay off, the warm-start time must stay small: the accelerator runs at every warm-start iteration, so its input must be cheap to compute and have a fixed dimension regardless of problem size. We consider LPs in the bounded form

$$\min_x \; c^\top x \quad \text{s.t.} \quad Gx \geq h, \quad Ax = b, \quad \ell \leq x \leq u, \tag{2}$$

where the bounds ℓ and u may contain infinite entries. PDHG (Chambolle & Pock, 2011) is a classical first-order method for Equation (2). Let $\mathcal{K} = [G; A] \in \mathbb{R}^{m \times n}$, $q = [h; b]$, and $\mathcal{Y} =$

$\mathbb{R}^{m_1}_+ \times \mathbb{R}^{m_2}$ for the m_1 inequality and m_2 equality rows, with $m = m_1 + m_2$. PDHG alternates

$$x_{k+1} = \mathrm{proj}_{[\ell,u]}\big(x_k - \tau_k(c - \mathcal{K}^\top y_k)\big), \quad y_{k+1} = \mathrm{proj}_{\mathcal{Y}}\big(y_k + \sigma_k(q - \mathcal{K}(2x_{k+1} - x_k))\big), \quad (3)$$

where $\mathrm{proj}_{\mathcal{C}}(v) := \arg\min_{w\in\mathcal{C}} \frac{1}{2}\|w - v\|_2^2$. PDHG-based solvers such as PDLP first go through an oscillatory exploration phase and only then converge toward a solution (Applegate et al., 2023). This observation motivates a lightweight step-size predictor that adjusts τ_k and σ_k at every iteration to shorten this early phase. After K warm-start iterations, the accelerator hands the warm start to the solver, which continues until it reaches ϵ. The best number of warm-start iterations $K_i^\star = \arg\min_K \mathcal{T}_i(\theta, K; \epsilon)$, balances the two terms and depends on both the instance and ϵ. However, $K_i^\star$ is known only in hindsight, since evaluating it requires running solver to termination from the warm start at every candidate K. A practical accelerator therefore needs an online rule that selects K_i.

Formally, AHEAD keeps Equation (3) and only rescales its step sizes:

$$(\tilde\tau_k, \tilde\sigma_k) = \begin{cases} (u_k^{\mathrm{p}}\tau_k,\ u_k^{\mathrm{d}}\sigma_k), & k < K_i, \\ (\tau_k,\ \sigma_k), & k \geq K_i, \end{cases} \quad (4)$$

where τ_k, σ_k are the solver's own step sizes and $u_k^{\mathrm{p}}, u_k^{\mathrm{d}} > 0$ are learned step-size multipliers. AHEAD thus makes two online decisions: it predicts $(u_k^{\mathrm{p}}, u_k^{\mathrm{d}})$ at every warm-start iteration and selects K_i for each instance and ϵ. The predictor outputs only two scalars per iteration, and the iterate at every warm-start iteration remains a valid PDHG iterate.

3.2 Weight-Shared Step-Size Predictor

To predict $(u_k^{\mathrm{p}}, u_k^{\mathrm{d}})$ at each warm-start iteration, we design a weight-shared step-size predictor (Figure 1, left). It maps the current solver state to the two multipliers and shares its weights across iterations. We first describe the state and then the predictor.

Solver state. At each warm-start iteration k, the predictor reads a state $\mathbf{s}_k$ that describes the current progress of PDHG. To choose the step sizes for the next iteration, the state must answer four questions: how much the current iterate violates the primal and dual constraints, whether a larger step would remain stable, how to split the step between primal and dual, and whether recent steps make progress or oscillate. PDLP (Applegate et al., 2026) already answers each question with its own mechanism: its termination residuals, its step-acceptance check, its primal weight, and its restart rule (Appendix A). We therefore build one group of features from each mechanism, reusing only the solver's own intermediate results, so that $\mathbf{s}_k$ is cheap to compute and has the same dimension for every LP, as required in Section 3.1:

$$\mathbf{s}_k = \Big[\ \underbrace{\bar r_k^{\mathrm{p}}, \bar r_k^{\mathrm{d}}}_{\text{feasibility}},\ \underbrace{\mathcal{M}_k, \mathcal{N}_k, \mathcal{U}_k, \mathcal{S}_k}_{\text{local stability}},\ \underbrace{\bar\omega_k}_{\text{primal–dual scaling}},\ \underbrace{C_k^x, C_k^y, \rho_k}_{\text{local trajectory}}\ \Big] \in \mathbb{R}^{10}. \quad (5)$$

Feasibility. A primal–dual pair (x, y) of an LP is optimal if and only if x is primal feasible, y is dual feasible, and the duality gap is zero. The first group tracks the two feasibility conditions: the primal residual $\bar r_k^{\mathrm{p}} = \log\big(1 + \|Ax_k - b\|_2/\sqrt{m}\big)$ measures how much x_k violates the primal constraints, and the dual residual $\bar r_k^{\mathrm{d}} = \log\big(1 + \|c - A^\top y_k\|_2/\sqrt{n}\big)$ measures how much y_k violates the dual constraints. Large residuals indicate an early phase of the solve, where larger steps pay off, and small residuals call for finer steps.

Local stability. The second group tells the predictor whether a larger step would remain stable. With constant step size, PDHG converges when $\tau\sigma\|\mathcal{K}\|_2^2 < 1$ (Chambolle & Pock, 2011, Theorem 1). This bound is often too conservative in practice. PDLP (Applegate et al., 2021) instead adapts the step size with a local *stability condition* on the actual movement: it computes the update in Equation (3) with a trial step size, keeps the update if the condition holds, and otherwise shrinks the step size and recomputes the update (Appendix A). Let $\Delta x_k = x_k - x_{k-1}$ and $\Delta y_k = y_k - y_{k-1}$ denote the change between the last two iterations, and write $\tau_k = \eta_k/\omega_k$ and $\sigma_k = \eta_k\omega_k$, where $\eta_k > 0$ sets the overall step size and $\omega_k > 0$ is the solver's primal weight. With the weighted movement $\mathcal{M}_k = \frac{\omega_k}{2}\|\Delta x_k\|_2^2 + \frac{1}{2\omega_k}\|\Delta y_k\|_2^2$ and the primal–dual interaction $\mathcal{N}_k = \langle A\Delta x_k, \Delta y_k\rangle$, PDLP's condition can be written as

$$\mathcal{M}_k/\eta_k - \mathcal{N}_k \geq 0. \quad (6)$$

This condition always holds when $\mathcal{N}_k \leq 0$. When $\mathcal{N}_k > 0$, it limits the step size to $\eta_k \leq \mathcal{M}_k/\mathcal{N}_k$. We define the utilization as the ratio of the current step size to this upper limit, $\mathcal{U}_k = \eta_k/(\mathcal{M}_k/\mathcal{N}_k) = \eta_k\mathcal{N}_k/\mathcal{M}_k$, so the condition holds exactly when $\mathcal{U}_k \leq 1$. We therefore include $\mathcal{M}_k, \mathcal{N}_k, \mathcal{U}_k$, and the bounded ratio $\mathcal{S}_k = \tanh(\mathcal{N}_k/\mathcal{M}_k)$ in the state: $\mathcal{U}_k$ tells the predictor how close the current step size is to this limit, and the sign of $\mathcal{S}_k$ tells it whether the condition can be violated at all. The predictor thus observes the same condition that the classical solver imposes on each update, and can set the next multipliers.

Primal–dual scaling. The third group tells the predictor how to split the step between primal and dual. The solver sets this split through its primal weight: $\tau_k = \eta_k/\omega_k$ and $\sigma_k = \eta_k\omega_k$. The best pair of multipliers thus depend on ω_k, so we include the detrended log-weight $\bar{\omega}_k = \log\omega_k - 0.8\log(n/m)$, which treats reciprocal primal and dual scalings symmetrically.

Local trajectory. The last group tells the predictor whether recent steps make progress or oscillate. PDHG iterates often spiral around saddle points before settling, which slows early progress (Chambolle & Pock, 2011; Applegate et al., 2023). PDLP counters this with restarts triggered by the normalized duality gap, which measures how far an iterate is from the saddle point but needs an extra optimization problem to compute, so PDLP evaluates it only periodically (Applegate et al., 2021). However, our predictor needs this information at every iteration, and computing the gap that often would be too expensive. We therefore measure the directional consistency of consecutive updates with cosine similarities, which are inexpensive to compute at every iteration:

$$C_k^x = \frac{\langle \Delta x_k, \Delta x_{k-1}\rangle}{\|\Delta x_k\|_2\|\Delta x_{k-1}\|_2 + \delta}, \qquad C_k^y = \frac{\langle \Delta y_k, \Delta y_{k-1}\rangle}{\|\Delta y_k\|_2\|\Delta y_{k-1}\|_2 + \delta}, \tag{7}$$

where the small constant $\delta > 0$ prevents division by zero. Up to δ, each cosine equals $+1$ when consecutive updates point in the same direction and -1 when they reverse, and it does not change when either update is rescaled, for example by a new step size. Values near $+1$ thus indicate steady progress and suggest that a larger step is safe, whereas values near -1 indicate oscillation and need to adapt the step smaller. We also include the movement ratio $\rho_k = \log(\|\Delta z_k\|_2/\|\Delta z_{k-1}\|_2)$ with $\Delta z_k = (\Delta x_k, \Delta y_k)$, which shows whether the updates are growing ($\rho_k > 0$) or shrinking ($\rho_k < 0$). These features give the predictor the progress information that PDLP's restart rule relies on. Thus, AHEAD itself never restarts during the warm-start phase.

Weight-shared step-size predictor. Given $\mathbf{s}_k$, the predictor outputs the logarithms of the two step-size multipliers,

$$\left[\ell_k^{\mathrm{p}},\, \ell_k^{\mathrm{d}}\right]^\top = f_\theta(\mathbf{s}_k), \qquad u_k^{\mathrm{p}} = e^{\ell_k^{\mathrm{p}}}, \quad u_k^{\mathrm{d}} = e^{\ell_k^{\mathrm{d}}}, \tag{8}$$

where f_θ is a small MLP with tanh activations. Predicting logarithms keeps both multipliers positive, and Equation (4) applies them to the solver's native step sizes.

The primal–dual update directions of the solver are faithfully kept in AHEAD, as we find them to be efficient and reliable directions towards optimality. The primal and dual step sizes are much harder to tune, since first-order methods like PDLP lack the higher-order information as interior-point methods to find the optimal step size for each update. Hand-designed rules either overestimate or underestimate the optimal step size. Learning the optimal step sizes only can thus reap the most benefit of learning without introducing more model complexity that lowers per-iteration efficiency.

Another core design principle is a looped architecture for recursive solution refinement: we apply the same predictor f_θ at every warm-start iteration. This is a paradigm shift from existing neural solvers with a fixed model depth and independent model parameters per-layer: they require more parameters and cannot adapt to the varying number of iterations. AHEAD runs like a classical solver: it recursively refines primal-dual solutions, without a pre-determined number of iterations, until the handoff policy ends the warm-start phase. This adaptive recursion is key to generalizability across instances with different sizes and difficulties, as reported in other literature such as (Chen et al., 2020) and recurrent networks (Schwarzschild et al., 2021).

3.3 Learned Handoff Policy

With the weight-shared predictor, AHEAD can run for any number of warm-start iterations. For moderate accuracy requirements, such as $\epsilon = 10^{-4}$, the warm-start iterations alone already bring

the solution close to the accuracy. Because AHEAD is fully aligned with the solver, it enables a second capability: when higher accuracy is required, AHEAD hands its solution directly to the solver as warm start. This warm start substantially reduces the number of iterations the solver needs to converge at high accuracy, while still achieving a large end-to-end speedup over a cold start.

A key question in this handoff is when to stop the warm-start phase and switch to the solver's own iterations. Switching too late wastes warm-start iterations that no longer reduce the remaining solver time, whereas switching too early gives the solver a weaker warm start than AHEAD could provide. We address this with a learned handoff policy g_φ (Figure 1, right). It selects the number of warm-start iterations K_i for any given target accuracy ϵ. Since the best choice $K_i^\star$ is known only in hindsight (Section 3.1), the policy makes this decision online. At every warm-start iteration k, it reads a *handoff state* $\mathbf{h}_k$, predicts the end-to-end time of stopping now and of continuing, and ends the warm-start phase once stopping is predicted to be faster. Unlike halting in adaptive neural computation (Graves, 2016; Banino et al., 2021), this decision depends on the remaining solver time after handoff, not on the quality of a network output.

Handoff state. Whether to handoff depends on how close the solver is to convergence and how fast it is approaching it. The solver monitors the primal residual, dual residual, and duality gap,

$$e_k^{\mathrm{p}} = \frac{\|Ax_k - b\|_2}{1 + \|b\|_2}, \quad e_k^{\mathrm{d}} = \frac{\|c - A^\top y_k\|_2}{1 + \|c\|_2}, \quad e_k^{\mathrm{g}} = \frac{|c^\top x_k - b^\top y_k|}{1 + |c^\top x_k| + |b^\top y_k|}, \tag{9}$$

and reaches target accuracy ϵ when $\max\{e_k^{\mathrm{p}}, e_k^{\mathrm{d}}, e_k^{\mathrm{g}}\} \le \epsilon$ (Applegate et al., 2021). Convergence is decided by the slowest criterion, and the same residual may be close enough for a loose ϵ but far from a tight one. We therefore measure, for each criterion, the log-gap to the target accuracy

$$\Delta_k^j(\epsilon) = \log \frac{e_k^j + \delta}{\epsilon + \delta}, \qquad j \in \{\mathrm{p}, \mathrm{d}, \mathrm{g}\}, \tag{10}$$

where $\delta > 0$ avoids $\log 0$. Because ϵ is considered in the handoff state, the same learned handoff policy can adapt the number of warm-start iterations across target accuracies.

Handoff rule. At warm-start iteration k, AHEAD can either hand off now or continue. An MLP g_φ maps the handoff state $\mathbf{h}_k$ to

$$\left[\widehat{Q}_{\mathrm{handoff}}(\mathbf{h}_k),\ \widehat{Q}_{\mathrm{continue}}(\mathbf{h}_k)\right] = g_\varphi(\mathbf{h}_k), \tag{11}$$

where $\widehat{Q}_{\mathrm{handoff}}$ and $\widehat{Q}_{\mathrm{continue}}$ estimate the end-to-end time of handing off now and of handing off at the best later iteration, respectively. Both are divided by the cold-start time, so they are comparable across instances of different sizes and difficulty. AHEAD hands off at the first warm-start iteration k with $\widehat{Q}_{\mathrm{handoff}}(\mathbf{h}_k) \le \widehat{Q}_{\mathrm{continue}}(\mathbf{h}_k)$, or after $K_{\max}$ iterations if this condition never holds. estimate the same quantity, so the rule needs no extra threshold. After the handoff, the solver continues from the warm start with its own step-size rule and termination criterion, so every returned solution meets the target accuracy ϵ, exactly as a cold-start solve does.

3.4 Multi-Horizon Training

To serve different accuracy requirements, we train the predictor over varying horizons K rather than a single fixed one. The handoff policy may end learned acceleration at any checkpoint, so the predictor must leave a useful iterate at every candidate horizon. Formally, let $\mathcal{H}$ be the set of candidate horizons and $(\mathbf{x}_i^\star, \mathbf{y}_i^\star)$ an optimal primal–dual solution of instance i. We train the predictor by

$$\min_\theta \ \mathbb{E}_{i\sim\mathcal{D}}\, \mathbb{E}_{K\sim\mathcal{U}(\mathcal{H})} \left[\frac{\|\hat{\mathbf{x}}_i^K - \mathbf{x}_i^\star\|_2^2}{\|\mathbf{x}_i^\star\|_2^2 + \delta} + \frac{\|\hat{\mathbf{y}}_i^K - \mathbf{y}_i^\star\|_2^2}{\|\mathbf{y}_i^\star\|_2^2 + \delta} \right], \tag{12}$$

where $(\hat{\mathbf{x}}_i^K, \hat{\mathbf{y}}_i^K)$ is the handoff iterate after K learned iterations. The normalized distance to the optimum serves as a differentiable proxy for the remaining solver work T_{solver}. This objective can be viewed as training over a distribution of horizons rather than a single fixed depth. By sampling K, the predictor learns a state-dependent rule that remains useful for any horizon the handoff policy may pick, and the shared weights let it run for horizons outside $\mathcal{H}$. We initialize the output layer of f_θ at zero, so that $(u_k^{\mathrm{p}}, u_k^{\mathrm{d}}) = (1, 1)$ and training starts from the solver's native step-size rule.

Training the handoff policy. Unlike the distance used to train the predictor, the end-to-end time is measured in wall-clock and is not differentiable. We therefore freeze the trained predictor, roll it out on each training instance, and measure at every warm-start iteration k the end-to-end time of handing off at k, divided by the cold-start time, $C_i(k) = \mathcal{T}_i(\theta, k; \epsilon) \,/\, \mathcal{T}_{i,\mathrm{cold}}$, using actual solver runs from the warm start rather than a surrogate. The regression targets for the two predictions of the handoff policy are

$$Q_{i,\mathrm{handoff}}(k) = C_i(k), \qquad Q_{i,\mathrm{continue}}(k) = \min_{k<j\le K_{\max}} C_i(j), \tag{13}$$

the normalized end-to-end time of handing off now and the best one attainable by handing off later. We fit g_φ by

$$\min_{\varphi} \frac{1}{|\mathcal{D}_{\mathrm{handoff}}|} \sum_{i\in\mathcal{D}_{\mathrm{handoff}}} \frac{1}{|\mathcal{J}_i|} \sum_{k\in\mathcal{J}_i} \sum_{a\in\{\mathrm{handoff},\,\mathrm{continue}\}} \big(\widehat{Q}_a(\mathbf{h}_{i,k}) - Q_{i,a}(k)\big)^2, \tag{14}$$

where $\mathcal{D}_{\mathrm{handoff}}$ is the set of training instances for the handoff policy and $\mathcal{J}_i$ is the set of warm-start iterations evaluated on instance i. Because both outputs are regressed onto the same normalized end-to-end time, the handoff rule compares them directly, without a hand-tuned threshold.

4 Experiments

In this section, we first examine the proposed adaptive accelerator against common baselines, and further examine the generalization capabilities, adaptive stopping and difficulty. Finally, we conduct ablation analysis to analyze the underlying structure of AHEAD.

4.1 Experimental Settings

Datasets and metrics. We evaluate on two LP families: synthetic packing LPs with $n \in [10^5, 10^6]$ nonnegative variables, one equality constraint, and about $0.1n$ sparse inequalities, and PageRank LPs on Barabási–Albert graphs (Applegate et al., 2021; Iouditski & Nesterov, 2014). Both follow the LP form in Section 3.1. We report the end-to-end speedup, whose wall-clock time includes AHEAD's inference, presolve, and solver execution, and the reduction in iterations, both at the same target accuracy ($\epsilon = 10^{-4}$ by default). Test instances are disjoint from the training instances. Appendices C.1, C.2, and C.3 give the generation details and the measurement protocol.

Baselines and Evaluation. We compare against three groups of baselines. Cold-start solvers form the first group: Gurobi GPU PDHG (`PDHGGPU=1`) and OR-Tools PDLP. The second group is the learned baseline PDHG-Net (Li et al., 2024), the closest learned method to ours. The third group contains two variants of AHEAD: a fixed horizon selected on validation, and a per-instance optimal $K_i^\star$ that marks the best achievable handoff.

Training Details. The candidate horizons are $\mathcal{H} = \{8, 16, 32, 64, 128\}$. We train the predictor for 200 epochs with Adam, a learning rate of 10^{-3}, cosine annealing, and a batch size of 8. Each mini-batch shares one sampled horizon K, which enables batched rollouts and still gives an unbiased estimate of Equation 12. We clip gradients so that pathological primal or dual errors do not dominate an update. The handoff policy is then trained on the frozen predictor's trajectories.

4.2 Acceleration

Table 1 compares AHEAD with PDHG-Net and with a fixed number of warm-start iterations selected on validation, all on the Gurobi GPU-PDHG backbone at $\epsilon = 10^{-4}$.

The test sets contain 50 held-out packing LPs, 20 larger packing LPs with $n \in [10^6, 10^7]$, and 20 packing LPs from an unseen distribution. (1) AHEAD generalizes across scale and distribution. Its predictor only rescales step sizes from a fixed-dimensional solver state and never predicts the solution, unlike PDHG-Net, whose predictions are tied to the training distribution. (2) Adaptive handoff nearly matches the best handoff. The best number of warm-start iterations varies across instances, so a fixed number wastes iterations or stops too early, whereas the handoff policy stops once further iterations no longer save time. (3) The advantage grows with problem size. Larger PageRank LPs need more iterations, and the weight-shared predictor runs as many as each instance needs, whereas PDHG-Net has a fixed number of layers (Figure 2).

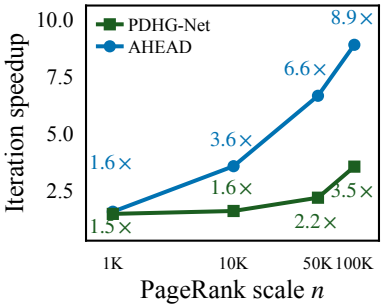
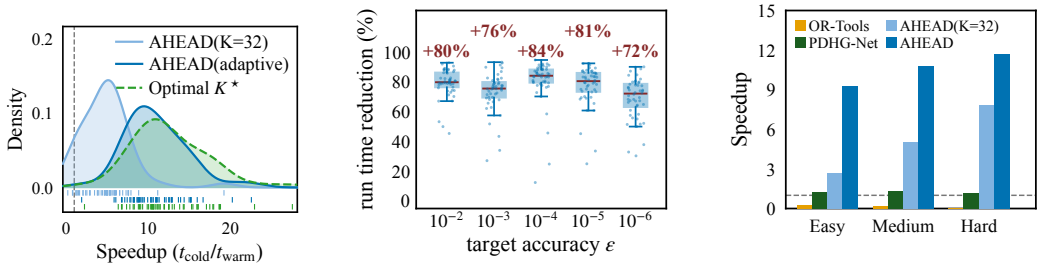


Figure 2: Speedup on PageRank LPs.

Table 1: Acceleration with the Gurobi PDHG backbone on packing-type LPs.

Method	**IID** (n=50)		**Size shift** (n=20)		**Unseen distribution** (n=20)	
	Speedup	Time (s)	Speedup	Time (s)	Speedup	Time (s)
Cold PDHG (Gurobi)	1.00×	18.2	1.00×	5.3	1.00×	3.2
Best validation K (K=32)	5.63×	3.2	5.09×	1.1	3.25×	0.99
PDHG-Net (Li et al., 2024)	1.23×	14.8	1.18×	4.5	1.06×	3.1
AHEAD	**11.07×**	**1.6**	**11.59×**	**0.46**	**6.48×**	**0.50**
Per-instance optimal $K^\star$	12.36×	1.5	12.75×	0.42	7.06×	0.46

4.3 Generalization



(a) Fixed vs. adaptive handoff. (b) Robustness across target acc. (c) Speedup across difficulty tiers.

Figure 3: Robustness across acceleration horizons, target accuracies, and problem difficulty levels.

Does AHEAD rely on solver continuation? We disable handoff on a representative LP instance and compare AHEAD's residual trajectory with adaptive AHEAD, cold-start PDHG, and PDHG-Net. Figure 4 supports two observations. (1) AHEAD can reach the target tolerance without native continuation. When AHEAD runs without handoff, it reduces the primal residual and reaches the target, whereas PDHG-Net remains above it. (2) Learned acceleration and solver refinement are complementary. Adaptive handoff reaches the target earlier than AHEAD without handoff or the cold-start solver, indicating that AHEAD accelerates early trajectory while the native solver completes the refinement. Handoff improves efficiency rather than providing progress that AHEAD cannot achieve on its own.

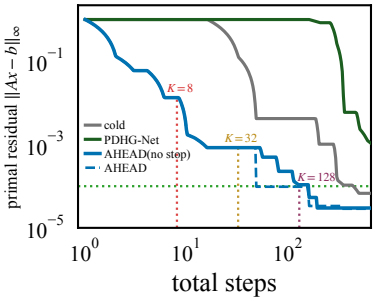


Figure 4: Primal-residual trajectories without early solver handoff.

Adaptive Stopping. To evaluate whether AHEAD can select an effective number of warm-start iterations, we compare adaptive handoff with a fixed $K = 32$ and the per-instance optimal $K^\star$, the

$K \in \{1, \ldots, 128\}$ with the shortest end-to-end time on each instance; we also evaluate robustness across target accuracies. The figures support two observations. (1) Adaptive handoff comes close to the per-instance optimal $K^\star$. In Figure 3(a), the adaptive speedup distribution shifts markedly beyond the fixed-horizon baseline and toward the oracle, showing that the current solver state provides useful information for selecting an instance-specific handoff point. (2) Acceleration persists across target accuracies. Figure 3(b) shows consistently large runtime reductions across all evaluated tolerances. Because every run must satisfy the same PDLP termination criterion, these gains demonstrate consistent speedups for convergence without relaxing final solution accuracy.

Difficulty. To test whether acceleration persists as instances become harder, we generate 60 packing-LP test instances unseen during training and divide them into three equal-sized groups, Easy, Medium, and Hard, based on their cold-start Gurobi PDHG runtimes. We then split them into three equal-sized groups: the shortest third as Easy, the middle third as Medium, and the longest third as Hard. Figure 3(c) reports AHEAD achieves approximately $9\times$, $11\times$, and $12\times$ speedup on easy, medium, and hard instances, respectively. The corresponding fixed $K = 32$ results are roughly $3\times$, $5\times$, and $8\times$. Adaptive handoff improves speedup in every tier, while OR-Tools PDLP and PDHG-Net remain below the cold Gurobi baseline in this comparison.

4.4 Ablation

Which solver-state features matter? Table 2 incrementally introduces the four groups of solver-state features. Adding the primal and dual residuals reduces the total number of steps from 104.4 to 92.5. Movement and nonlinearity provide the largest incremental improvement, further reducing the total to 80.4 steps. Incorporating the primal weight and cosine features subsequently reduces the total to 79.3 and 76.8, respectively. Overall, the complete representation achieves a 26.4% reduction relative to the fixed step-size baseline. These results indicate that feasibility, local stability, primal-dual scaling, and trajectory geometry provide complementary information for step-size prediction.

Table 2: Incremental ablation of solver-state features. Each row retains all features introduced above it.

Input state	**Total steps** ↓	**Δ** (%) ↑
Fixed stepsize	104.4	–
+ Residuals	92.5	11.4
+ Movement/nonlinearity	80.4	**13.1**
+ Primal weight ω_k	79.3	1.4
+ Cosine geometry (AHEAD)	**76.8**	3.2

Does the improvement come from model capacity? Table 3 compares controller architectures while fixing the solver-state representation and training objective. Increasing the controller from five layers with width 64 to seven layers with width 128 increases the total number of steps from 68.5 to 72.9. The improvement therefore cannot be explained simply by increased network capacity. In contrast, sharing controller parameters across iterations produces the best result, requiring 67.8 total steps and 35.8 continuation steps. This supports matching the controller parameterization to the iteration-invariant update structure of the underlying optimizer.

Table 3: Predictor-architecture ablation on the evaluation set.

Controller	**Total steps** ↓	**Cont. steps** ↓
5 layers, width 64	68.5	36.5
7 layers, width 128	72.9	40.9
Weight-shared (AHEAD)	**67.8**	**35.8**

5 Conclusion

We asked what neural accelerator design achieves end-to-end speedups on large-scale LPs across sizes and target accuracies. AHEAD answers with three design choices. First, it learns only the step sizes and keeps the solver's update directions, so each warm-start iteration stays inexpensive and every iterate remains a valid warm start. Second, its weight-shared predictor runs recurrently, so the warm-start phase can last any number of iterations. Third, a learned handoff policy selects this number for each instance and accuracy. Together, these choices let the solver reach the target accuracy in fewer iterations with little time per iteration, yielding many-fold end-to-end speedups on packing and PageRank LPs across unseen scales, difficulty levels, and target accuracies. Future work will extend AHEAD to real-world LPs and other first-order solvers.

AI USE STATEMENT

In this work, we used ChatGPT to identify and improve issues related to grammar, wording, clarity, and readability in the manuscript. ChatGPT was not used to develop the research ideas or methodology, formulate mathematical claims or proofs, design or conduct experiments, implement the method, generate or process data, or analyze or interpret results. All technical content, experiments, results, and conclusions were produced by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a general-purpose method for accelerating linear programming solvers. It does not involve human subjects, personal or sensitive information, or private datasets. All experiments use synthetically generated packing and PageRank linear programs following the procedures described in the paper. We are not aware of any direct privacy, fairness, security, or societal risks arising from this work, although the broader impact of general-purpose optimization technology may depend on its downstream application.

REPRODUCIBILITY STATEMENT

We provide the information needed to reproduce AHEAD and its evaluation throughout the paper and supplementary material. Section 3 specifies the solver state, the predictor and the handoff policy, the termination criterion, the training objectives, and the two-stage training procedure. Section 4.1 describes the datasets, baselines, evaluation metrics, and training hyperparameters. The appendix provides the PDLP mechanisms that AHEAD builds on (Appendix A), the training algorithm (Algorithm 1), the PageRank formulation (Appendix C.1), the synthetic packing-LP generation and data splits (Appendix C.2), and the hardware, software, and timing protocol (Appendix C.3). Training, validation, and test instances are generated using disjoint random seeds and checked for duplicate content. The anonymous repository accompanying this submission provides the source code, model configurations, instance-generation scripts, and experiment commands required to reproduce the reported results.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29, pp. 3981–3989, 2016. URL https://arxiv.org/abs/1606.04474.

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

David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1–2): 133–184, 2023. doi: 10.1007/s10107-022-01901-9. URL https://doi.org/10.1007/s10107-022-01901-9.

David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. PDLP: A practical first-order method for large-scale linear programming. *Mathematical Programming Computation*, 2026. doi: 10.1007/s12532-026-00309-2. URL https://doi.org/10.1007/s12532-026-00309-2.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. doi: 10.48550/arXiv.2107.05407. URL https://arxiv.org/abs/2107.05407. Presented at the 8th ICML Workshop on Automated Machine Learning.

Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40(1):120–145, 2011.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022. URL https://jmlr.org/papers/v23/21-0308.html.

Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1520–1530. PMLR, 2020. URL https://proceedings.mlr.press/v119/chen20c.html.

George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

Zhenan Fan, Xinglu Wang, Oleksandr Yakovenko, Abdullah Ali Sivas, Owen Ren, Yong Zhang, and Zirui Zhou. Smart initial basis selection for linear programs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 9650–9664. PMLR, 2023. URL https://proceedings.mlr.press/v202/fan23d.html.

Xi Gao, Jinxin Xiong, Akang Wang, Qihong Duan, Jiang Xue, and Qingjiang Shi. IPM-LSTM: A learning-based interior point method for solving nonlinear programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 122891–122916, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/de0da9c42ee713f2ceaeed7bc40c522d-Abstract-Conference.html.

Tom Goldstein, Min Li, Xiaoming Yuan, Ernie Esser, and Richard Baraniuk. Adaptive primal-dual hybrid gradient methods for saddle-point problems. *arXiv preprint arXiv:1305.0546*, 2013.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. doi: 10.48550/arXiv.1603.08983. URL https://arxiv.org/abs/1603.08983.

Anatoli Iouditski and Yuri Nesterov. Primal-dual subgradient methods for minimizing uniformly convex functions, 2014. URL https://arxiv.org/abs/1401.1792.

Tomoharu Iwata and Shinsaku Sakaue. Learning to generate projections for reducing dimensionality of heterogeneous linear programming problems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 26627–26641. PMLR, 2025.

Yufei Kuang, Xijun Li, Jie Wang, Fangzhou Zhu, Meng Lu, Zhihai Wang, Jia Zeng, Houqiang Li, Yongdong Zhang, and Feng Wu. Accelerate presolve in large-scale linear programming via reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47 (8):6660–6672, 2025. doi: 10.1109/TPAMI.2025.3562286. URL https://doi.org/10.1109/TPAMI.2025.3562286.

Bingheng Li, Linxin Yang, Yupeng Chen, Senmiao Wang, Haitao Mao, Qian Chen, Yao Ma, Akang Wang, Tian Ding, Jiliang Tang, and Ruoyu Sun. PDHG-unrolled learning-to-optimize method for large-scale linear programming. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 29164–29180. PMLR, 2024. URL https://proceedings.mlr.press/v235/li24ce.html.

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016. doi: 10.48550/arXiv.1606.01885. URL https://arxiv.org/abs/1606.01885.

Xijun Li, Qingyu Qu, Fangzhou Zhu, Jia Zeng, Mingxuan Yuan, Kun Mao, and Jie Wang. Learning to reformulate for linear programming. *arXiv preprint arXiv:2201.06216*, 2022. doi: 10.48550/arXiv.2201.06216. URL https://arxiv.org/abs/2201.06216.

Tianyi Lin, Shiqian Ma, Yinyu Ye, and Shuzhong Zhang. An ADMM-based interior-point method for large-scale linear programming. *Optimization Methods and Software*, 36(2–3):389–424, 2021. doi: 10.1080/10556788.2020.1821200.

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

Tianhao Liu, Shanwen Pu, Dongdong Ge, and Yinyu Ye. Learning to pivot as a smart expert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8073–8081, 2024. doi: 10.1609/aaai.v38i8.28646.

Anqi Lu and Junchi Yan. Learning initial basis selection for linear programming via duality-inspired tripartite graph representation and comprehensive supervision. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40831–40847. PMLR, 2025.

Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. *Operations Research*, 73(6):3440–3452, 2025. doi: 10.1287/opre.2024.1069. URL https://doi.org/10.1287/opre.2024.1069.

Yura Malitsky and Thomas Pock. A first-order primal-dual algorithm with linesearch. *SIAM Journal on Optimization*, 28(1):411–432, 2018. doi: 10.1137/16M1092015. URL https://doi.org/10.1137/16M1092015.

Yurii Nesterov. Subgradient methods for huge-scale optimization problems. *Mathematical Programming*, 146(1–2):275–297, 2014. doi: 10.1007/s10107-013-0686-4.

Alex Oshin, Rahul Ghosh, Augustinos Saravanos, and Evangelos Theodorou. Deep FlexQP: Accelerated nonlinear programming via deep unfolding. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/84cd19dedba8b8b7461d86c246f2c372-Abstract-Conference.html.

Chendi Qian, Didier Chételat, and Christopher Morris. Exploring the power of graph neural networks in solving linear optimization problems. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pp. 1432–1440. PMLR, 2024. URL https://proceedings.mlr.press/v238/qian24a.html.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024. URL https://jmlr.org/papers/v25/23-1174.html.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks, 2021. URL https://arxiv.org/abs/2106.04537.

Michael Sucker and Peter Ochs. A generalization result for convergence in learning-to-optimize. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57212–57230. PMLR, 2025.

Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3751–3760. PMLR, 2017. URL https://proceedings.mlr.press/v70/wichrowska17a.html.

A ENGINEERING ACCELERATIONS IN PDLP

Classical PDHG uses fixed primal and dual step sizes chosen according to a global stability condition, making its convergence highly sensitive to conservative step-size selection and mismatched primal–dual scaling. Beyond presolving and diagonal preconditioning, PDLP accelerates conventional PDHG through three coupled mechanisms: **adaptive step-size selection**, **adaptive restarting**, and **dynamic primal weighting**.

First, PDHG parameterizes its primal and dual step sizes as

$$\tau_k = \frac{\eta_k}{\omega_k}, \quad \sigma_k = \eta_k \omega_k$$

where η_k controls the overall step magnitude and ω_k controls the relative primal-dual scaling. Conventional PDHG selects η_k according to the global condition $\eta_k \leq 1/\|A\|_2$, which guarantees convergence but is often overly conservative. PDLP instead evaluates each candidate step using the local condition $\eta_k \leq \bar{\eta}_k := \frac{\|z_{k+1}-z_k\|_{\omega_k}^2}{2(y_{k+1}-y_k)^\top A(x_{k+1}-x_k)}$, where $z_k = (x_k, y_k)$ and $\|z\|_\omega^2 = \omega\|x\|_2^2 + \frac{1}{\omega}\|y\|_2^2$. The next candidate step size is updated as $\eta_{k+1} = \min\left\{\left(1-(k+1)^{-0.3}\right)\bar{\eta}_k, \left(1+(k+1)^{-0.6}\right)\eta_k\right\}$. A candidate violating the local condition is rejected and recomputed using a smaller step. Because $\bar{\eta}_k$ depends on the realized primal-dual movement rather than only on the global operator norm, PDLP can take substantially larger steps along locally stable portions of the trajectory.

The second mechanism **controls trajectory history** through adaptive restarting. Within outer iteration n, PDLP maintains a step-size-weighted average of the inner iterates, $\bar{z}^{n,t+1} = \frac{\sum_{i=1}^{t+1}\eta^{n,i}z^{n,i}}{\sum_{i=1}^{t+1}\eta^{n,i}}$. Progress is evaluated using the normalized duality gap

$$\rho_r^n(z) = \frac{1}{r}\max_{\substack{\hat{z}=(\hat{x},\hat{y})\in\mathcal{Z} \\ \|\hat{z}-z\|_{\omega^n}\leq r}} \{\mathcal{L}(x,\hat{y}) - \mathcal{L}(\hat{x},y)\},$$

and $\mu_n(z, z_{\text{ref}}) = \rho_{\|z-z_{\text{ref}}\|_{\omega^n}}(z)$.

At each restart check, PDLP selects between the current iterate and its weighted average:

$$z_c^{n,t+1} = \begin{cases} z^{n,t+1}, & \mu_n\left(z^{n,t+1}, z^{n,0}\right) < \mu_n\left(\bar{z}^{n,t+1}, z^{n,0}\right) \\ \bar{z}^{n,t+1}, & \text{otherwise} \end{cases}$$

The outer loop is restarted when the normalized duality gap exhibits sufficient decay, when local progress stalls after necessary decay has been achieved, or when the inner loop becomes excessively long. This mechanism retains averaging when it is beneficial but discards accumulated trajectory history once it ceases to improve the normalized gap.

Finally, PDLP updates the **primal weight** only at restart points. Let $\Delta_x^n = \left\|x^{n,0} - x^{n-1,0}\right\|_2$, $\Delta_y^n = \left\|y^{n,0} - y^{n-1,0}\right\|_2$. The new primal weight is obtained through log-space smoothing: $\omega^n = \exp\left(\theta \log\frac{\Delta_y^n}{\Delta_x^n} + (1-\theta)\log\omega^{n-1}\right)$, where PDLP uses $\theta = 0.5$. This update approximately balances the weighted primal and dual distances while avoiding abrupt changes that may destabilize PDHG. Consequently, the three mechanisms address distinct limitations of conventional PDHG: adaptive step sizes control the step magnitude using local stability, restarts manage accumulated trajectory history, and the primal weight balances progress between the primal and dual spaces.

B TRAINING

C DATASET

C.1 PAGERANK

Following Applegate et al. (2021), we generate PageRank LPs with the formulation of Iouditski & Nesterov (2014), using a damping factor $\lambda = 0.85$. The resulting LP has one equality row $\mathbf{1}^\top x = 1$

Algorithm 1 Random-Horizon Training

Require: Dataset $\mathcal{D}$ with targets $(\mathbf{x}^\star, \mathbf{y}^\star)$; $\mathcal{K} = \{8, 16, 32, 64, 128\}$; batch size B

Initialization
repeat
  Sample a mini-batch $\mathcal{B} \subset \mathcal{D}$ and a shared $K \sim \mathrm{Uniform}(\mathcal{K})$
  for all $i \in \mathcal{B}$ **do**
    $(\hat{\mathbf{x}}_i^K, \hat{\mathbf{y}}_i^K) \leftarrow \mathrm{Rollout}_K(f_\theta, i)$
    $d_i \leftarrow \dfrac{\|\hat{\mathbf{x}}_i^K - \mathbf{x}_i^\star\|_2^2}{\|\mathbf{x}_i^\star\|_2^2 + \delta} + \dfrac{\|\hat{\mathbf{y}}_i^K - \mathbf{y}_i^\star\|_2^2}{\|\mathbf{y}_i^\star\|_2^2 + \delta}$
  end for
  $\mathcal{L}_{10^{-8}} \leftarrow \dfrac{1}{B} \sum_{i \in \mathcal{B}} d_i$
  Update θ using $\nabla_\theta \mathcal{L}_{10^{-8}}$
until convergence

and n inequality rows, and matches the LP form in Section 3.1 with $S = 1$ and the transition constraints written as sparse linear inequalities.

For each instance, we first sample a Barabási–Albert preferential-attachment graph with attachment degree 3. Let $S' \in \mathbb{R}^{n \times n}$ denote its column-normalized adjacency matrix, such that S' is column stochastic. Given a damping factor $\lambda \in (0, 1)$, the PageRank transition matrix is

$$S = \lambda S' + \frac{1 - \lambda}{n} \mathbf{1}\mathbf{1}^\top. \tag{15}$$

The corresponding PageRank vector can be obtained as a feasible solution to

$$\begin{aligned} \text{find} \quad & x, \\ \text{subject to} \quad & Sx \le x, \\ & \mathbf{1}^\top x = 1, \\ & x \ge 0. \end{aligned} \tag{16}$$

Directly constructing S would introduce a dense teleportation matrix. Using the normalization constraint $\mathbf{1}^\top x = 1$, we instead retain the equivalent sparse constraints

$$\lambda (S'x)_i + \frac{1 - \lambda}{n} \le x_i, \qquad i = 1, \dots, n. \tag{17}$$

C.2 Synthetic Packing LPs

We construct a family of synthetic packing LPs of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & \mathbf{1}^\top x = S, \\ & Ax \le b, \\ & x \ge 0. \end{aligned} \tag{18}$$

Each instance has $n \in [10^5, 10^6]$ variables and $m = 0.1n$ inequality constraints. The constraint matrix is generated as

$$A_{ij} = M_{ij} \widetilde{A}_{ij}, \qquad M_{ij} \sim \mathrm{Bernoulli}(\rho), \qquad \widetilde{A}_{ij} \sim \mathrm{Unif}(0.5, 1.0), \tag{19}$$

with density $\rho = 100/n$, so each inequality involves about 100 variables on average. The variables have no explicit upper bounds, and the instances match the LP form in Section 3.1 directly. The objective coefficients are sampled as $c_j \sim \mathrm{Unif}(-1, 2)$.

For each instance, we first sample the equality right-hand side as $S \sim \mathrm{Unif}(0.3, 0.7)$. We then draw

$$u \sim \mathrm{Dirichlet}(\mathbf{1}_n), \qquad x_0 = Su, \tag{20}$$

such that $x_0 \geq 0$ and $\mathbf{1}^\top x_0 = S$. The inequality right-hand side is constructed as

$$b = Ax_0 + s, \tag{21}$$

where the entries of s are sampled independently. This construction makes x_0 feasible and controls instance difficulty through the magnitude of the available slack: larger slack values produce mostly inactive inequalities, whereas smaller values yield a larger fraction of binding constraints at the optimum.

Data splits. The training instances and the in-distribution test set are drawn from the generator above with $n \in [10^5, 10^6]$. The size-shift test set contains 20 instances with $n \in [10^6, 10^7]$ from the same generator. The distribution-shift test set contains 20 instances whose generation parameters differ from those used for training. The difficulty tiers in Figure 3(c) group 60 held-out instances by their cold-start time. Training, validation, and test instances are generated using disjoint random seeds and checked for duplicate content. For the difficulty study (Figure 3(c)), we sort all test instances by their cold-start time on Gurobi PDHG and split them into three equal-sized groups: the shortest third as Easy, the middle third as Medium, and the longest third as Hard.

C.3 Experimental Environment and Measurement Protocol

All experiments were conducted on the same machine. Table 4 summarizes the hardware and software environment.

Table 4: Experimental environment.

Component	Configuration
GPU	NVIDIA A100 PCIe 40GB; driver 580.173.02
CPU	2× Intel Xeon Gold 5317; 24 cores, 48 threads
Memory	251 GiB
OS	Ubuntu 24.04.3 LTS; Linux 6.8.0-139
Python	3.12.9
PyTorch	2.9.0
CUDA toolkit	12.0 (V12.0.140)
OR-Tools	9.15.6755
Gurobi	13.0.3, GPU-enabled build

Timing metric. Unless stated otherwise, the aggregate end-to-end speedup is

$$\text{Speedup} = \frac{\sum_i \mathcal{T}_{i,\text{cold}}}{\sum_i \mathcal{T}_i(\theta, K_i; \epsilon)}, \tag{22}$$

where $\mathcal{T}_i = T_{\text{warm}} + T_{\text{solver}}$ is the end-to-end time defined in Section 3.

Warm-start iterations and handoff. The number of warm-start iterations K counts complete PDHG updates whose step-size multipliers are predicted by AHEAD. After K iterations, AHEAD passes the current primal iterate $\hat{\mathbf{x}}_K$ and dual iterate $\hat{\mathbf{y}}_K$ to the solver as its initial primal and dual solution; for Gurobi, these are assigned through `PStart` and `DStart`. No predictor state or step size is transferred, and the solver continues with its own update rule. The warm-start iterations are included in the end-to-end time. The fixed number of warm-start iterations used as a baseline is selected on validation data.