CLOSED-LOOP PRECONDITIONING FOR ANISOTROPIC DIFFUSION VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Preconditioners are commonly held fixed throughout a linear solve, but the evolving residual provides feedback on the effectiveness of earlier corrections. We investigate whether learning to exploit this feedback can improve convergence on strongly anisotropic diffusion problems. Our Closed-Loop Preconditioner (CLP) uses reinforcement learning to incrementally update an explicit preconditioner within first-order Richardson iteration. A permutation-equivariant graph policy updates a diagonal-plus-low-rank preconditioner that is symmetric positive definite by construction, with analytic Galerkin weighting of the learned subspace. Controlled comparisons with learned static and open-loop policies support the benefit of solver feedback. CLP reduces iteration counts relative to algebraic multigrid-preconditioned Richardson and achieves shorter measured solver times than the tested classical baselines. The policy transfers from short training trajectories on small systems to substantially larger systems and supports longer solves through defect-correction cycles, without retraining.

1 INTRODUCTION

Solving linear systems is a recurring task in scientific computing, and the effectiveness of iterative methods depends strongly on preconditioning (Benzi, 2002; Saad, 2003). Strongly anisotropic diffusion, arising for example in field-aligned transport in magnetized plasmas, provides a particularly challenging regime: transport is concentrated along spatially varying field directions, producing highly directional couplings that can make effective fixed preconditioning difficult and lead to slow iterative convergence.

A preconditioner is commonly constructed before iteration and held fixed throughout the solve, preventing it from responding to information revealed as the solve evolves. Classical adaptive methods already allow preconditioners and correction spaces to vary during a solve, but their adaptation is typically governed by prescribed numerical rules (Saad, 1993; Morgan, 2002). This suggests learning the adaptation rule itself from the evolving solve. Existing learned approaches have primarily focused on constructing a preconditioner (Li et al., 2023; Häusner et al., 2024) or solver setup before iteration, or on producing learned corrections within the iterative solver (Chen, 2025; Rudikov et al., 2024). We instead learn a reusable update rule that incrementally updates an explicit preconditioner state, conditioned on the current residual and iterate. We refer to this feedback-conditioned formulation as *closed-loop preconditioning*.

Each preconditioner update affects both the current correction and future solver states, making closed-loop preconditioning a sequential decision problem. We therefore use reinforcement learning (RL) to optimize this rule directly from solver trajectories. Residual reduction provides the learning signal without differentiating through the numerical iteration or precomputing target preconditioners or inverse actions.

To instantiate this formulation, we introduce the **C**losed-**L**oop **P**reconditioner (CLP), a learned, structure-preserving preconditioner updated throughout the solve. The policy learns how the preconditioner adapts, while analytic Galerkin weighting determines the correction within the learned subspace, yielding an SPD preconditioner by construction.

We study this approach within first-order Richardson iteration to isolate the contribution of preconditioner adaptation. Focusing on strongly anisotropic diffusion, we make the following contributions:

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

- We formulate closed-loop preconditioning as a sequential decision problem, learning a reusable rule that adapts an explicit preconditioner in response to the evolving solver state. Closed-loop adaptation achieves better convergence than matched learned static and open-loop policies, demonstrating the benefit of solver feedback (Table 2).
- We introduce a structure-preserving realization that combines learned preconditioner adaptation with analytic Galerkin weighting, yielding an SPD preconditioner by construction. Canonicalization and permutation equivariance further encode the scaling and relabelling symmetries of the linear system.
- We show that the resulting policy generalizes well beyond its training conditions, transferring from short trajectories on small systems to substantially larger systems and longer solves without retraining. Across these transferred settings, CLP requires fewer iterations and achieves lower measured solver time than the tested classically preconditioned Richardson baselines (Figures 1 and 2).

2 Related Work

Classical and adaptive preconditioning. Classical preconditioners include diagonal scaling, incomplete factorizations, domain decomposition and multigrid (Benzi, 2002; Saad, 2003; Ruge & Stüben, 1987). Beyond fixed constructions, deflation and augmentation introduce correction subspaces (Nabben & Vuik, 2004; Gaul et al., 2013), while recycling and low-rank updates reuse information across related systems (Parks et al., 2006; Bergamaschi, 2020). Flexible Krylov methods further allow the preconditioner itself to vary across iterations (Saad, 1993; Notay, 2000; Simoncini & Szyld, 2002). Together, these methods establish that preconditioners and correction subspaces need not remain fixed, but their adaptation is governed by prescribed numerical rules rather than a learned update rule conditioned on the evolving solver state.

Learning preconditioners and solver setup. Learning has been used to construct preconditioners through approximate factorizations (Li et al., 2023; Häusner et al., 2024), learned corrections to classical preconditioners (Trifonov et al., 2026), and sparse approximate inverses (Li et al., 2025). Learning has also been applied to classical solver setup, including multigrid prolongation operators (Greenfeld et al., 2019; Luz et al., 2020), parameter selection (Antonietti et al., 2023), and graph coarsening (Taghibakhshi et al., 2021). Although the learned objects differ, these approaches construct a preconditioner or solver setup before the iterative solve and then hold it fixed during that solve. They therefore learn how to construct a preconditioner, rather than how it should evolve in response to the evolving solver state.

Learned iterative corrections and subspaces. Beyond constructing a preconditioner or solver setup before the solve, learning can also act within the iteration itself. Learned solvers and nonlinear preconditioners use neural models to provide corrections within the iterative solve (Hsieh et al., 2019; Ackmann et al., 2020; Chen, 2025; Rudikov et al., 2024). Learning has also been used to construct coarse or deflation spaces, including Galerkin corrections (Kopaničáková & Karniadakis, 2025; Kopaničáková et al., 2025). These approaches operate within the solve, but typically use the learned model to supply a per-iteration correction or construct a learned subspace. Our approach instead maintains and updates an explicit preconditioner state, incrementally modifying the diagonal and low-rank basis across iterations while retaining analytic Galerkin weighting.

Feedback-driven and online solver adaptation. Learning has also been used for feedback-driven adaptation in numerical solvers and optimization. PEARL formulates preconditioner construction as a contextual bandit, using actor-critic learning to construct a preconditioner in a single decision (Millard et al., 2025). For sequences of related linear systems, bandit methods have been used to adapt solver parameters and preconditioner configurations from feedback across successive solves (Khodak et al., 2024; 2026). Related work in optimization adapts gradient scaling online from feedback along the optimization trajectory (Gao et al., 2025). Together, these works show that online feedback can guide the adaptation of solver configurations and iterative updates, but the adaptation is either made in a single decision, occurs across a sequence of solves, or targets optimization parameters rather than an accumulated preconditioner within a single solve. CLP instead learns a closed-loop update rule that uses the evolving solver state to incrementally modify a structured SPD preconditioner across iterations.

3 PRELIMINARIES

Problem setting. We consider linear systems

$$Ax = b, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite (SPD), and $x, b \in \mathbb{R}^n$ denote the unknown and right-hand-side vectors, respectively. Such systems arise, for example, from discretizations of elliptic PDEs (Brenner & Scott, 2007; LeVeque, 2007; Strang & Fix, 2008). Our formulation applies to general SPD operators, with a particular focus on strongly anisotropic diffusion motivated by field-aligned transport in magnetized plasmas.

Richardson iteration. We solve these systems using Richardson iteration (Varga, 1999; Barrett et al., 1994),

$$x_t = x_{t-1} + \alpha P_t^{-1} r_{t-1}, \qquad r_t = b - Ax_t, \tag{2}$$

where t is the iteration index, x_t is the current iterate, and r_t is the corresponding residual. We initialize $x_0 = 0$, so that $r_0 = b$. The step size $\alpha > 0$ is held fixed within each solve, while the preconditioner P_t may vary across iterations. Although we refer to P_t as the preconditioner, our method parameterizes and applies P_t^{-1} directly and never forms P_t.

Richardson iteration provides a simple setting for studying adaptive preconditioning. Each step applies the current preconditioner directly to the residual, without momentum or Krylov-subspace acceleration. In methods such as CG and GMRES (Hestenes & Stiefel, 1952; Saad & Schultz, 1986), convergence depends jointly on the preconditioner and the Krylov-subspace dynamics, making the effect of preconditioner updates harder to isolate. We therefore use Richardson iteration throughout to compare preconditioning strategies within a common solver framework.

Error propagation. For the unique solution $x^\star$, let $e_t = x_t - x^\star$. Equation 2 gives

$$e_t = (I - \alpha P_t^{-1} A)e_{t-1}, \qquad e_T = (I - \alpha P_T^{-1} A) \cdots (I - \alpha P_1^{-1} A)e_0. \tag{3}$$

For a fixed preconditioner, this becomes repeated application of a single amplification matrix, whose spectrum characterizes asymptotic convergence. When P_t varies, however, convergence depends on the full operator product, not the spectrum of any single step. This motivates adapting the preconditioner to the evolving solver state, so later updates can account for the effects of earlier iterations.

Closed-loop preconditioning. Consequently, we study closed-loop preconditioning, where each preconditioner update can depend on the current solver state. We therefore view the solve as a sequential decision process, where the policy updates the preconditioner P_t from information available at each iteration. We seek a shared policy across systems that achieves a small final relative residual $\|r_T\|_2/\|r_0\|_2$ after a horizon of T steps. Section 4 specifies the policy and reinforcement learning objective based on logarithmic residual reduction.

4 METHOD

The previous section framed closed-loop preconditioning as a sequential decision process in which an explicit preconditioner state is updated from the evolving solver state. We now instantiate this formulation in four steps. We first define a structured preconditioner parametrization that remains SPD throughout the solve. We then formulate its incremental updates as a reinforcement learning problem and parameterize the resulting update rule with a graph policy whose inductive biases reflect the structure of A. Finally, we show how the learned short-horizon policy can be reused through defect-correction cycles to support substantially longer solves without retraining.

4.1 ADAPTIVE LOW-RANK PRECONDITIONER

We seek a compact adaptive preconditioner whose learned updates remain numerically structured. For the symmetric positive definite (SPD) systems considered here, we require P_t^{-1} to preserve positive definiteness throughout the solve. CLP therefore combines a strictly positive diagonal component with a low-rank correction:

$$P_t^{-1} = D_t^{-1} + U_t C_t U_t^\top. \tag{4}$$

Here, $D_t^{-1} = \mathrm{diag}(\exp(d_t) + \varepsilon)$ is the diagonal component, with $d_t \in \mathbb{R}^n$ parameterized in log space and $\varepsilon > 0$ providing numerical stabilization. The matrix $U_t \in \mathbb{R}^{n\times k}$, with $k \ll n$, defines a learned low-rank subspace, while $C_t \in \mathbb{R}^{k\times k}$ weights the correction within that subspace. The policy updates d_t and U_t at every iteration, while C_t is determined analytically from the current subspace. Specifically, when $U_t^\top A U_t$ is nonsingular,

$$C_t = (U_t^\top A U_t)^{-1}. \tag{5}$$

In practice, we add a small diagonal regularization before inversion so that the weighting remains well defined when U_t is rank deficient. Applied to the residual, the unregularized term $U_t C_t U_t^\top r_{t-1}$ gives the Galerkin correction over $\mathrm{span}(U_t)$, minimizing the quadratic energy over updates in the learned subspace (Nabben & Vuik, 2004; Gaul et al., 2013). Thus, the policy learns the correction subspace, while its weighting is determined analytically by Galerkin projection.

This construction has three useful properties. First, for the unregularized Galerkin form in Equation 5, the correction depends only on the learned subspace rather than on its particular basis. For any invertible $M \in \mathbb{R}^{k\times k}$,

$$U_t M \left((U_t M)^\top A\, U_t M\right)^{-1} (U_t M)^\top = U_t (U_t^\top A U_t)^{-1} U_t^\top. \tag{6}$$

The weighting also has the correct inverse scaling: $A \mapsto cA$ gives $C_t \mapsto c^{-1} C_t$. Second, the implemented construction preserves positive definiteness. Since $A \succ 0$, the regularized Galerkin matrix is positive definite, making the low-rank term positive semidefinite. Together with the strictly positive diagonal component, this guarantees $P_t^{-1} \succ 0$ at every iteration regardless of the policy output. This positive definiteness ensures that $P_t^{-1} r_{t-1}$ is a descent direction for the quadratic energy $\frac{1}{2} x^\top A x - b^\top x$ whenever $r_{t-1} \neq 0$. Third, the correction remains computationally tractable for $k \ll n$. Forming $U_t^\top A U_t$ requires k matrix-vector products and the assembly of a small $k \times k$ Gram matrix, followed by a $k \times k$ factorization.

Section 5.4 compares alternative low-rank weightings, while Appendix B.4 specifies these alternatives and examines the preconditioner components and rank.

4.2 Reinforcement Learning Formulation

Having defined the preconditioner state, we next learn how it should evolve from solver feedback. We treat each solve as one episode of a finite-horizon Markov decision process of length T. At iteration t, the environment state consists of the solver state (x_t, r_t), preconditioner state (d_t, U_t), and the fixed system (A, b) throughout the episode. The policy receives an observation derived from this environment state, as specified in Section 4.3.

Actions. The policy outputs an increment to the preconditioner state,

$$a_t = (\Delta d_t, \Delta U_t), \qquad d_{t+1} = d_t + \Delta d_t, \qquad U_{t+1} = U_t + \Delta U_t. \tag{7}$$

We initialize $d_0 = 0$ and $U_0 = 0$. Predicting increments rather than absolute parameters allows the policy to refine an accumulated preconditioner state across iterations, rather than reconstructing it from scratch at every step. Given the action a_t, the transition is deterministic: the preconditioner state is updated as above, the Galerkin weighting is recomputed from the updated basis (Equation 5), and the updated parameters define P_{t+1}^{-1} through Equation 4. The solver then applies P_{t+1}^{-1} to r_t to obtain the next iterate (Equation 2).

Rewards. We use the logarithmic residual reduction achieved by each solver step,

$$R_t = \log \frac{\|r_t\|_2}{\|r_{t+1}\|_2}. \tag{8}$$

With $\gamma = 1$, the cumulative reward telescopes to $\sum_{t=0}^{T-1} R_t = \log(\|r_0\|_2 / \|r_T\|_2)$, so maximizing return is equivalent to minimizing the final logarithmic relative residual, while providing a learning signal at every iteration. The reward is computed directly from quantities available during the solve, requiring neither the exact solution $x^\star$ nor a converged reference solve.

Learning algorithm. We optimize the policy with proximal policy optimization (PPO) (Schulman et al., 2017), using solver trajectories directly without differentiating through the numerical iteration.

Each training episode samples a new operator and right-hand side from the target problem family, so that a single update rule is learned across systems. We train over a fixed horizon T, which limits rollout cost and the temporal span of credit assignment. The learned policy is then applied to unseen systems with fixed weights; its reuse beyond the training horizon is described in Section 4.4. Action processing, numerical stabilization, and optimization settings are specified in Appendix A.2.

4.3 Policy Architecture and Inductive Biases

Having specified the closed-loop learning problem, we parameterize the update rule with a graph neural network defined on the structure of A. The policy canonicalizes solver-state observations and maps them to preconditioner updates in a permutation-equivariant manner.

Graph representation and observations. We represent A by its sparsity graph: each unknown is a node, each nonzero off-diagonal entry A_{ij} defines an undirected edge with feature $|A_{ij}|$, and $|A_{ii}|$ is represented by a self-loop. At decision step t, node i has raw features

$$o_{t,i} = \big[\, (r_t)_i,\ (x_t)_i,\ (b)_i,\ t/T \,\big]. \tag{9}$$

The accumulated preconditioner state (d_t, U_t) is not observed directly.

Canonicalization. We canonicalize the raw observations to remove dependence on equivalent coordinate changes and overall vector magnitude. First, we express the operator and solver state in a Jacobi-scaled frame. Let $s = \sqrt{|\operatorname{diag}(A)|}$; the node features become r_t/s, $s \odot x_t$, and b/s, while the edge features become $|A_{ij}|/(s_i s_j)$. These quantities are invariant under the equivalent change of variables $A \mapsto \Lambda A \Lambda$, $b \mapsto \Lambda b$, and $x_t \mapsto \Lambda^{-1} x_t$ for any positive diagonal Λ. The diagonal preconditioner is decoded as $\exp(d_t)/|\operatorname{diag}(A)| + \varepsilon$, so $d_0 = 0$ and $U_0 = 0$ recover Jacobi preconditioning up to numerical stabilization. Second, we normalize each observation vector in the Jacobi frame by its Euclidean norm. This removes sensitivity to a common positive rescaling of b, x_t, and r_t, up to numerical stabilization, and reduces sensitivity to the changing right-hand-side magnitude when the policy is reused across defect-correction cycles (Section 4.4).

Graph policy and equivariance. After canonicalization, L rounds of message passing aggregate information from each node's L-hop neighborhood. Each node then outputs $k+1$ action coordinates: one diagonal increment and one row of the low-rank basis increment, which together form Δd_t and ΔU_t. Shared node-wise transformations and permutation-invariant neighbor aggregation make the policy permutation equivariant: relabelling the unknowns consistently relabels the preconditioner update. The critic pools node representations to produce a permutation-invariant scalar. Parameter sharing across nodes also allows the same policy to operate at different system sizes. Appendix B.3 gives the formal equivariance statement and an empirical consistency test.

4.4 Deployment Beyond the Training Horizon

The policy is trained over a fixed horizon of T decisions, but reaching a prescribed residual tolerance may require substantially longer solves. Rather than extrapolating the policy beyond its training horizon, we reuse the learned update rule through successive defect-correction cycles.

Given the current iterate x, we form $r = b - Ax$ and approximately solve the correction equation $Ae = r$. Each cycle starts from $e_0 = 0$, resets the preconditioner state to $d_0 = 0$ and $U_0 = 0$, and restarts the iteration index at $t = 0$ within each cycle. We then apply up to T learned Richardson steps to the correction equation. Within each cycle, the preconditioner evolves closed-loop exactly as during training. At iteration t within a cycle, the policy observes the correction residual $r - Ae_t$, iterate e_t, and right-hand side r, using the same canonicalization as in Section 4.3. The resulting correction is added to the current solution, $x \leftarrow x + e$, and the process repeats with fixed policy weights. The correction residual satisfies

$$r - Ae_t = b - A(x + e_t), \tag{10}$$

so each cycle continues to reduce the residual of the original system. Convergence is therefore monitored using $\|b - A(x + e_t)\|_2/\|b\|_2$, and cycles repeat until the prescribed tolerance is reached or the iteration budget is exhausted. Resetting the iteration index and preconditioner state at the start of each cycle keeps the policy within its training horizon, while the observation normalization of Section 4.3 reduces sensitivity to the changing correction right-hand side. This allows the same short-horizon policy to support substantially longer solves without retraining.

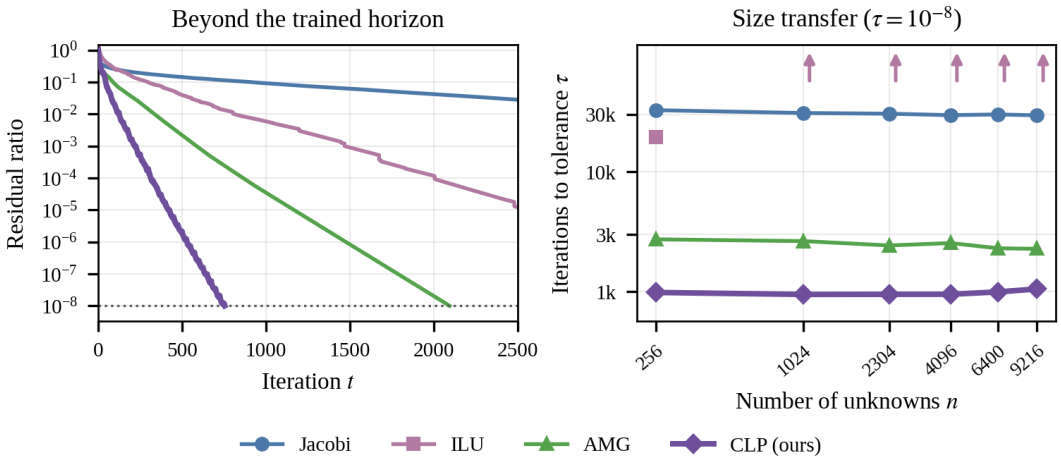

Figure 1: **Long-horizon reuse and system-size transfer.** CLP achieves the fastest residual reduction beyond the trained horizon and requires the fewest iterations to reach the target residual across all tested problem sizes. The policy is reused without retraining. Arrows indicate values beyond the plot range.

5 Experiments

Our experiments address four questions: **1)** whether CLP improves convergence and measured solver time over classical preconditioners on strongly anisotropic diffusion; **2)** whether policies trained on small systems over short trajectories transfer to larger systems and longer solves; **3)** how solver feedback, policy structure, and analytic subspace weighting contribute to performance; and **4)** how the advantage depends on anisotropy strength.

5.1 Experimental Setup

Unless otherwise stated, we consider magnetized diffusion with $n = 256$ unknowns and anisotropy ratio $\kappa_\parallel/\kappa_\perp = 10^4$. Policies are trained with PPO for 10,000 updates across five random seeds, using a horizon of $T = 25$ and a rank-4 Galerkin correction. We compare against unpreconditioned Richardson and Jacobi, incomplete LU (ILU) (Saad, 2003), and algebraic multigrid (AMG) preconditioning (Ruge & Stüben, 1987). All methods use Richardson iteration with separately tuned step sizes.

At the trained horizon, we report $\log_{10}(\|r_T\|_2/\|r_0\|_2)$; for longer solves, we measure iterations and solver time to a prescribed relative residual tolerance. Lower values indicate better performance. Unless otherwise stated, tables report instance-level medians $\pm$ half-IQR, averaged across training seeds for learned methods. Further experimental details are provided in Appendix A.

5.2 Generalization and Solver Efficiency

We first assess whether CLP improves convergence and measured solver time over classical preconditioners, and whether these gains persist on larger systems and longer solves. The trained policy is reused without retraining through the defect-correction procedure of Section 4.4.

Long-horizon reuse. Figure 1 (left) shows residual trajectories at $n = 256$, extending well beyond the training horizon. CLP maintains an approximately constant residual decay rate across successive defect-correction cycles, with its median relative residual reaching 10^{-8} in roughly one third as many iterations as AMG-preconditioned Richardson. The advantage therefore extends beyond the short-horizon objective used for training: repeated application of the same policy supports convergence to a stringent tolerance without learning a longer update schedule.

System-size transfer. We next apply the same policy to systems with up to $n = 9216$ unknowns, or 36 times the training size. Policy weights remain fixed, while Richardson step sizes are retuned for all methods at each size. Figure 1 (right) shows that CLP's iteration count changes only modestly across the tested sizes and remains below AMG's throughout. At the largest size, CLP reaches the tolerance in fewer than half as many iterations as AMG. The policy therefore retains its advantage

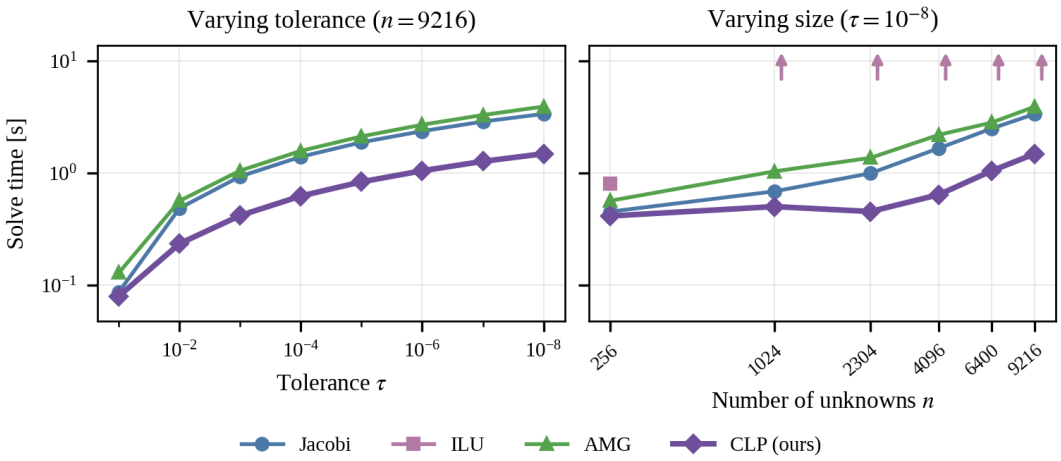

Figure 2: **Measured solver time.** At $n = 9216$, CLP reaches every residual tolerance faster than the baselines (left), and achieves the shortest time to 10^{-8} across all problem sizes (right). Arrows indicate values beyond the plot range or no finite median crossing within the evaluation budget.

when both system size and solve length increase, demonstrating that size transfer and reuse through defect correction remain effective in combination.

Measured solver time. Iteration savings need not translate into shorter solves, since CLP evaluates the policy and updates the preconditioner at every iteration. Figure 2 examines this trade-off across residual tolerances at $n = 9216$ (left) and across problem sizes at a tolerance of 10^{-8} (right). CLP achieves the lowest measured solver time across the displayed tolerances at the largest size and the shortest time to 10^{-8} at every tested size. At $n = 9216$, reaching 10^{-8} is approximately $2.6\times$ faster than AMG and $2.3\times$ faster than Jacobi. ILU is omitted from the tolerance sweep because it has no finite median crossing at the displayed tolerances.

Table 1 breaks down solver costs at $n = 9216$; reported times include correction-cycle initialization but exclude training and compilation. Policy evaluation and preconditioner updates account for most of CLP's per-iteration cost, making each step more expensive than Jacobi. The reduction in iteration count nevertheless compensates for this overhead. Relative to the tested AMG implementation, CLP benefits from both fewer iterations and lower setup and per-iteration costs. The measured solve-time advantage therefore reflects both improved convergence and the cost of the implemented operations.

Together, these results show that the same short-horizon policy transfers across both solve length and system size while retaining an end-to-end solver-time advantage.

5.3 Benefit of Solver Feedback

We now examine whether feedback from the evolving solver state contributes to CLP's performance. We compare CLP with two learned alternatives on magnetized diffusion at the trained horizon of $T = 25$.

Table 1: **Cost decomposition and time-to-tolerance.** At $n = 9216$, CLP reaches the target residual in the fewest iterations and with the shortest total solver time. Policy/iter includes observation construction, inference, state updates, and Galerkin refresh; Apply/iter includes preconditioner application, the Richardson update, and residual evaluation. Dashes indicate no finite median crossing.

	Cost [ms] (↓)			To 10^{-8} (↓)	
Method	Setup	Policy/iter	Apply/iter	Iters	Total Time [s]
Jacobi	0.44	0	0.114	29460	3.36
ILU	58.6	0	0.833	–	–
AMG	24.3	0	1.686	2270	3.91
CLP (ours)	6.76	1.330	0.050	**1051**	**1.49**

The *learned static* policy observes (A, b) at initialization and selects a single preconditioner, held fixed throughout the solve. The *open-loop* policy updates the preconditioner at every iteration using (A, b, t), but does not observe the residual r_t or iterate x_t. All three policies share the network architecture, preconditioner parametrization, training objective, and episode budget; their training procedures are detailed in Appendix A.2.

Table 2 shows that the open-loop policy improves only slightly on the learned static policy, while access to the evolving residual and iterate yields a larger improvement. These results support the benefit of solver feedback beyond time-dependent preconditioner updates alone under the tested training procedure.

The two learned alternatives isolate the role of solver feedback, while Figure 3 places them alongside the classical baselines to show how the resulting differences in convergence develop over the solve. AMG reduces the residual more rapidly during the initial iterations, whereas CLP sustains a steeper decay over the later iterations and achieves the lowest final residual among the methods shown. The advantage thus develops over the course of the solve as the policy repeatedly updates the preconditioner. The observation ablation in Appendix B.1 further examines the contribution of individual feedback channels, while the decision-interval sweep in Appendix B.2 shows weaker residual reduction as preconditioner updates become less frequent.

Table 2: **Ablation of solver feedback.** CLP achieves the lowest final residual, outperforming both learned static and open-loop preconditioning.

Method	$\log_{10}(\|r_T\|_2/\|r_0\|_2)$ (↓)
Learned static	-0.46 ± 0.04
Open-loop	-0.49 ± 0.04
CLP (ours)	$\mathbf{-0.72 \pm 0.13}$

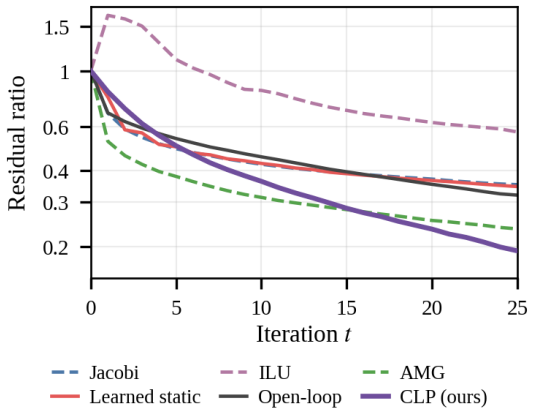

Figure 3: **Residual trajectories within the training horizon.** CLP achieves the lowest final residual and steepest late-stage decay among baselines.

5.4 Structured Policy and Preconditioner Design

Having established the benefit of solver feedback, we examine the architectural and numerical choices used to translate that feedback into preconditioner updates.

Canonicalization and operator structure. Table 3 compares the graph policy with an MLP that observes the solver state without the operator graph, each with and without canonicalization. All variants share the preconditioner parametrization, training objective, and episode budget. Removing canonicalization (Section 4.3) also changes the initial preconditioner from Jacobi to identity.

Table 3: **Ablation of the canonicalization and policy backbone.** The canonicalized graph policy achieves the lowest final residual.

Policy	$\log_{10}(\|r_T\|_2/\|r_0\|_2)$ (↓)
MLP w/o canon.	diverges
GNN w/o canon.	$+1.32 \pm 0.51$
MLP w/ canon.	-0.36 ± 0.04
GNN w/ canon. (CLP)	$\mathbf{-0.72 \pm 0.13}$

The canonicalized variants perform substantially better for both architectures, although this comparison also changes the initial preconditioner from identity to Jacobi. Among the canonicalized variants, the GNN substantially outperforms the MLP, supporting the benefit of the operator-aware graph representation. This benefit is achieved with a compact policy, indicating that the observed gains do not require a large network under the tested training budget (Appendix B.5).

Permutation consistency. Randomly permuting the unknowns leaves the graph policy's final residual unchanged up to small numerical differences, whereas the same transformation can turn residual reduction into residual growth for the MLP (Table 8). This illustrates the practical value of equivariance: changing the storage order of an otherwise identical system should not degrade solver performance.

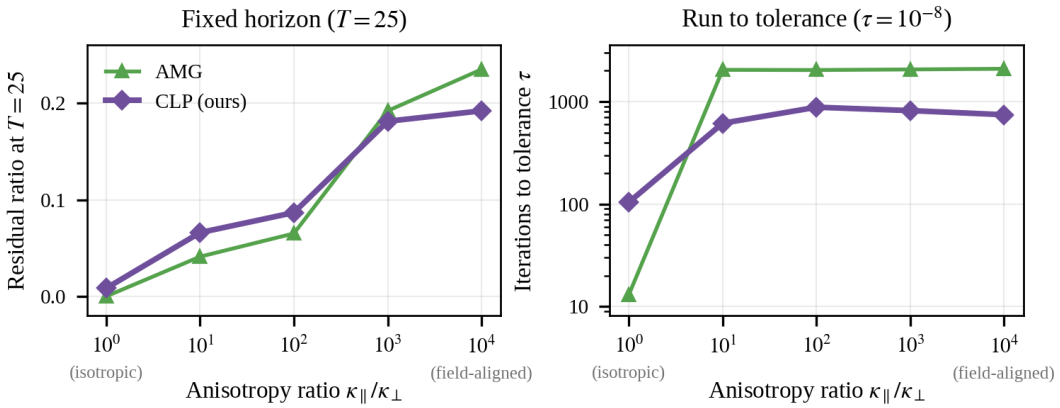

Figure 4: **Performance across anisotropy levels.** As anisotropy increases, CLP achieves lower final residuals than AMG at ratios 10^3 and 10^4 (left), and requires fewer iterations to reach 10^{-8} across ratios 10–10^4 (right). A separate CLP policy is trained at each anisotropy ratio.

Analytic subspace weighting. Table 4 compares Galerkin weighting with identity weights, learned positive diagonal weights, and Rayleigh-quotient weights, retaining rank 4, the graph backbone, and the training budget. Galerkin weighting achieves the strongest reported residual reduction. Unlike the alternatives, which weight basis directions independently, it incorporates their coupling through the full projected matrix $U_t^\top A U_t$. Appendix B.4 specifies the alternatives and provides additional component and rank ablations.

Table 4: **Ablation of low-rank weighting.** Galerkin gives the best result.

Weighting	$\log_{10}(\|r_T\|_2/\|r_0\|_2)$ (↓)
Identity	-0.44 ± 0.07
Learned	-0.59 ± 0.06
Rayleigh-quotient	-0.50 ± 0.04
Galerkin (CLP)	$\mathbf{-0.72 \pm 0.13}$

5.5 Effectiveness under Strong Anisotropy

We finally examine how CLP's advantage depends on the strength of anisotropy. We vary $\kappa_\parallel/\kappa_\perp$ from 1 to 10^4, training a separate policy and retuning Richardson step sizes at each ratio while retaining the other experimental settings.

Figure 4 (left) shows that AMG achieves the lowest residual at isotropy, but its advantage diminishes as anisotropy increases. CLP's performance degrades less across the sweep, yielding lower final residuals than AMG at ratios 10^3 and 10^4. The short-horizon comparison thus identifies strongly anisotropic diffusion as the regime where closed-loop preconditioning is most effective relative to this baseline.

At anisotropy ratios 10–10^4, the advantage is more pronounced when solving to a prescribed tolerance. Using the defect-correction procedure of Section 4.4, CLP reduces the median iteration count relative to AMG by a factor of approximately 2.3–3.3 when targeting a relative residual of 10^{-8} at anisotropy ratios from 10 to 10^4 (Figure 4, right). The benefit therefore extends beyond the fixed training horizon and appears over a broader range of anisotropy in longer solves.

6 Conclusion

We introduced CLP, a reinforcement-learning approach for closed-loop preconditioning that incrementally updates a structured SPD preconditioner from solver feedback. By combining learned adaptation with analytic Galerkin weighting, CLP maintains numerical structure while allowing the preconditioner to evolve throughout the solve. On strongly anisotropic diffusion, closed-loop adaptation outperforms matched learned static and open-loop alternatives, and the same policy transfers to larger systems and longer solves without retraining, reducing both iteration counts and measured solver time relative to the tested Richardson baselines. These results support closed-loop adaptation as a useful approach in regimes where fixed preconditioning remains difficult. Our study intentionally uses first-order Richardson iteration to isolate the effect of preconditioner adaptation; extending the approach to momentum and Krylov methods is a natural direction for future work.

Reproducibility Statement

We describe the experimental settings and implementation details to support reproducibility, and will release the source code alongside the paper upon acceptance. Section 4 specifies the preconditioner, policy architecture, and training objective. Appendix A details problem generation, training hyperparameters, baseline configurations, and evaluation and timing protocols. Appendix C specifies the supplementary problem distributions.

Ethics Statement

This work studies numerical methods for solving linear systems using synthetically generated problems. It involves no human subjects or personal data. We do not anticipate specific ethical concerns arising directly from the methods and experiments presented.

AI Use Statement

In this work, we used generative AI tools for implementing methods, refining hypotheses, providing feedback on research methodology and experiments, and interpreting results. We have not used generative AI tools for generating synthetic datasets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for mathematical proofs, assisting in writing proofs, translation, cleaning or reformatting datasets, or supporting qualitative and thematic data analysis. Additionally, we used generative AI tools to assist with code editing, figure preparation, literature review, and manuscript drafting and revision. We have reviewed all AI-assisted work. The authors checked AI-assisted analyses and interpretations against the experimental results and reviewed the implementation, figures, references, and manuscript for accuracy and consistency. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Jan Ackmann, Peter D. Düben, Tim N. Palmer, and Piotr K. Smolarkiewicz. Machine-learned preconditioners for linear solvers in geophysical fluid flows. *arXiv preprint arXiv:2010.02866*, 2020.

Paola F. Antonietti, Matteo Caldana, and Luca Dede'. Accelerating algebraic multigrid methods via artificial neural networks. *Vietnam Journal of Mathematics*, 51:1–36, 2023. doi: 10.1007/s10013-022-00597-w.

Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, 1994.

Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

Luca Bergamaschi. A survey of low-rank updates of preconditioners for sequences of symmetric linear systems. *Algorithms*, 13(4), 2020.

Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*, volume 15 of *Texts in Applied Mathematics*. Springer New York, 3 edition, 2007.

Jie Chen. Graph neural preconditioners for iterative solutions of sparse linear systems. In *International Conference on Learning Representations (ICLR)*, 2025.

Wenzhi Gao, Ya-Chi Chu, Yinyu Ye, and Madeleine Udell. Gradient methods with online scaling. In *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pp. 2192–2226. PMLR, 2025. URL https://proceedings.mlr.press/v291/gao25a.html.

André Gaul, Martin H Gutknecht, Jörg Liesen, and Reinhard Nabben. A framework for deflated and augmented Krylov subspace methods. *SIAM Journal on Matrix Analysis and Applications*, 34(2): 495–518, 2013.

Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. Learning to optimize multigrid PDE solvers. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

S. Günter, Q. Yu, J. Krüger, and K. Lackner. Modelling of heat transport in magnetised plasmas using non-aligned coordinates. *Journal of Computational Physics*, 209(1):354–370, 2005. doi: 10.1016/j.jcp.2005.03.021.

Paul Häusner, Ozan Öktem, and Jens Sjölund. Neural incomplete factorization: learning preconditioners for the conjugate gradient method. *Transactions on Machine Learning Research*, 2024.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. doi: 10.6028/jres.049.044.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural PDE solvers with convergence guarantees. In *International Conference on Learning Representations (ICLR)*, 2019.

Mikhail Khodak, Edmond Chow, Nina Balcan, and Ameet Talwalkar. Learning to relax: Setting solver parameters across a sequence of linear system instances. In *International Conference on Learning Representations*, 2024.

Mikhail Khodak, Min Ki Jung, Brian Wynne, Edmond Chow, and Egemen Kolemen. One-shot acceleration of transient PDE solvers via online-learned preconditioners. *Computer Physics Communications*, 327:110304, 2026. doi: 10.1016/j.cpc.2026.110304.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL `https://arxiv.org/abs/1412.6980`.

Alena Kopaničáková and George Em Karniadakis. DeepONet based preconditioning strategies for solving parametric linear systems of equations. *SIAM Journal on Scientific Computing*, 47(1): C151–C181, 2025.

Alena Kopaničáková, Youngkyu Lee, and George Em Karniadakis. Leveraging operator learning to accelerate convergence of the preconditioned conjugate gradient method. *Machine Learning for Computational Science and Engineering*, 1(2):39, 2025. doi: 10.1007/s44379-025-00039-7.

Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics, 2007.

Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. Learning preconditioners for conjugate gradient PDE solvers. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Zhehao Li, Kangbo Lyu, Yixuan Li, Tao Du, and Ligang Liu. Learning sparse approximate inverse preconditioners for conjugate gradient solvers on GPUs. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6489–6499. PMLR, 2020.

David Millard, Arielle Carr, Stéphane Gaudreault, and Ali Baheri. PEARL: Preconditioner enhancement through actor-critic reinforcement learning. *arXiv preprint arXiv:2501.10750*, 2025.

Ronald B. Morgan. GMRES with deflated restarting. *SIAM Journal on Scientific Computing*, 24(1): 20–37, 2002. doi: 10.1137/S1064827599364659.

R. Nabben and C. Vuik. A comparison of deflation and coarse grid correction applied to porous media flow. *SIAM Journal on Numerical Analysis*, 42(4):1631–1647, 2004.

Yvan Notay. Flexible conjugate gradients. *SIAM Journal on Scientific Computing*, 22(4):1444–1460, 2000.

Michael L. Parks, Eric de Sturler, Greg Mackey, Duane D. Johnson, and Spandan Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, 28(5): 1651–1674, 2006.

Alexander Rudikov, Vladimir Fanaskov, Ekaterina Muravleva, Yuri M. Laevsky, and Ivan Oseledets. Neural operators meet conjugate gradients: The FCG-NO method for efficient PDE solving. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 2024.

John W. Ruge and Klaus Stüben. Algebraic multigrid. In *Multigrid Methods*, chapter 4, pp. 73–130. SIAM, 1987.

Youcef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016. URL `https://arxiv.org/abs/1506.02438`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Valeria Simoncini and Daniel B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM Journal on Numerical Analysis*, 40(6):2219–2239, 2002.

Gilbert Strang and George Fix. *An Analysis of the Finite Element Methods, New Edition*. Wellesley-Cambridge Press, 2008.

Ali Taghibakhshi, Scott MacLachlan, Luke Olson, and Matthew West. Optimization-based algebraic multigrid coarsening using reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 12129–12140, 2021.

Vladislav Trifonov, Alexander Rudikov, Oleg Iliev, Yuri M. Laevsky, Ivan Oseledets, and Ekaterina Muravleva. Learning from linear algebra: A graph neural network approach to preconditioner design for conjugate gradient solvers. *Computational Methods in Applied Mathematics*, 26(2): 267–284, 2026. doi: 10.1515/cmam-2025-0170.

Richard S. Varga. *Matrix Iterative Analysis*, volume 27 of *Springer Series in Computational Mathematics*. Springer Berlin Heidelberg, 2 edition, 1999.

A Experimental Details

A.1 Magnetized Diffusion Problem Generation

This section specifies the magnetized diffusion distribution used in our main experiments. Unless otherwise stated, systems have $n = 256$ unknowns on an $n_x \times n_y$ periodic grid with $n_x = n_y = 16$ and $n = n_x n_y$. The distribution below describes the default setting; individual experiments change only the quantities explicitly identified.

We consider diffusion on the periodic unit square,

$$\frac{\partial f}{\partial \tau} = \nabla \cdot (\mathsf{D} \nabla f), \qquad \mathsf{D} = \kappa_\perp \left(I - vv^\top\right) + \kappa_\parallel vv^\top, \tag{11}$$

where f is the transported scalar and v is the spatially varying unit magnetic-field direction. The projections $vv^\top$ and $I - vv^\top$ select the directions parallel and perpendicular to the field, respectively. We use $\kappa_\parallel = 1$ and $\kappa_\perp = \varepsilon_\perp \kappa_\parallel$, with $\varepsilon_\perp = 10^{-4}$ by default, giving an anisotropy ratio $\kappa_\parallel / \kappa_\perp = 10^4$. Advection and antisymmetric cross-diffusion are excluded, yielding a symmetric diffusion operator.

The direction field is generated from a random periodic stream function,

$$\psi(\boldsymbol{\xi}) = \sum_{\boldsymbol{k} \in \mathcal{K}} \left[c_{\boldsymbol{k}} \cos(2\pi \boldsymbol{k} \cdot \boldsymbol{\xi}) + s_{\boldsymbol{k}} \sin(2\pi \boldsymbol{k} \cdot \boldsymbol{\xi}) \right], \qquad v = \frac{(-\partial_y \psi, \partial_x \psi)}{\|\nabla \psi\|_2}, \tag{12}$$

where $\boldsymbol{\xi} = (x, y)$ and the coefficients $c_{\boldsymbol{k}}, s_{\boldsymbol{k}}$ are independent standard-normal draws. The set $\mathcal{K}$ contains integer wavevectors satisfying $\|\boldsymbol{k}\|_\infty \le 2$ and either $k_y > 0$ or $(k_y = 0,\ k_x > 0)$. This selects one representative from each pair $\boldsymbol{k}, -\boldsymbol{k}$. The field directions are tangent to the level sets of ψ where $\nabla \psi \neq 0$.

The field is evaluated at cell centers on a square periodic grid with spacing $h = 1/n_x = 1/n_y$. To avoid normalizing a nearly vanishing gradient, a realization is accepted only if the smallest cell-center gradient norm is at least 10^{-3} times the largest. The sampler allows up to 50 attempts. Figure 5 illustrates the matrix structure and magnetic-field directions.

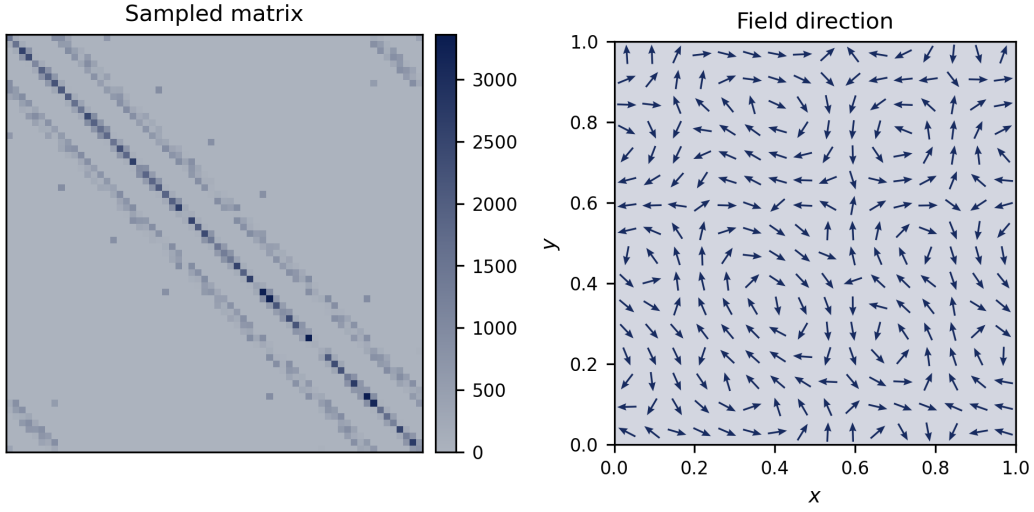

Figure 5: **Magnetized diffusion matrix and field visualization. Left:** entry magnitudes $|A_{ij}|$ of a sampled matrix at $n = 64$ (8×8 grid), shown at reduced size for readability. **Right:** sampled unit magnetic-field directions on the default 16×16 grid used in the experiments.

We use a corner-averaged discretization based on the symmetric scheme of Günter et al. (2005). A discrete gradient $G \in \mathbb{R}^{2n\times n}$ maps nodal values to cell-center gradients:

$$(Gf_h)_c = \frac{1}{2h}\begin{bmatrix} f_{10} - f_{00} + f_{11} - f_{01} \\ f_{01} - f_{00} + f_{11} - f_{10} \end{bmatrix}. \tag{13}$$

Here f_{ab} denotes the nodal value at $(x_i + ah, y_j + bh)$, for $a, b \in \{0, 1\}$, with periodic indexing. Writing $\Lambda = \mathrm{blockdiag}(\mathsf{D}_c)$, we discretize $-\nabla \cdot (\mathsf{D}\nabla)$ and construct the backward-Euler system matrix as

$$K = G^\top \Lambda G, \qquad A = I + \Delta\tau\, K, \qquad \Delta\tau = \frac{\eta h^2}{\kappa_\parallel}, \qquad \log\eta \sim \mathrm{Uniform}(\log 10^2, \log 10^4). \tag{14}$$

The dimensionless parameter η controls the strength of the diffusion term relative to the identity. The assembly produces a nine-point stencil and satisfies $K \succeq 0$, hence $A \succeq I$. On even periodic grids, the corner gradient annihilates both the constant and checkerboard modes; the identity term makes A nonsingular despite these null modes of K. When changing resolution, we keep the same Fourier mode set $\mathcal{K}$ and distribution of η, so the time step $\Delta\tau$ scales with h^2.

Right-hand sides and sampling. For each episode, a new matrix is sampled from the distribution above, and the right-hand side is drawn independently as $b \sim \mathcal{N}(0, I_n)$. Although the magnetized matrices arise from backward-Euler discretization, their right-hand sides in these experiments are Gaussian samples rather than states from a simulated diffusion trajectory.

A.2 Training Configuration

We use clipped PPO (Schulman et al., 2017) with generalized advantage estimation (GAE; Schulman et al., 2016) and Adam (Kingma & Ba, 2015). One PPO update consists of collecting a new rollout batch followed by three optimization epochs over that batch. Table 5 summarizes the default configuration. The observation construction and preconditioner parametrization are described in Section 4.

Architecture. The actor and critic have separate graph-network parameters. Each network consists of a node encoder followed by message-passing rounds. Messages are computed from the concatenated source and destination embeddings and the scalar edge weight, then averaged over incoming edges, including self-loops. A node-update network combines this average with the current embedding. The encoder, message network, and node-update network each contain two affine layers with $\tanh$ activations. Message and update parameters are shared across rounds. The actor uses separate linear heads for diagonal and basis updates. The critic averages the final node embeddings and applies a two-layer value head with one hidden $\tanh$ activation. Affine-layer weights and biases are initialized uniformly in $[-m^{-1/2}, m^{-1/2}]$, where m is the layer's input dimension.

Action processing and numerical details. The policy samples independent Gaussian coordinates with observation-dependent means and learned, observation-independent standard deviations. Samples are transformed by $\tanh$ and scaled to the increment bounds in Table 5; PPO log probabilities include the transformation Jacobian. The increments are added to the preconditioner state, which is then clipped entrywise to the accumulated-state bounds.

To distinguish stored and applied quantities, let V_t denote the stored canonical basis and let $U_t = S^{-1}V_t$ denote its decoded counterpart, where $S = \mathrm{diag}(s)$ is the Jacobi scaling from Section 4.3. This distinction is implementation-specific; for notational simplicity, the main text suppresses the canonical storage variable V_t and uses U_t for the low-rank basis throughout. The policy's basis increments and the entrywise bounds in Table 5 apply to V_t. The decoded basis U_t is used to construct and apply the low-rank correction. The Galerkin matrix is symmetrized numerically and regularized:

$$H_t = U_t^\top A U_t, \qquad C_t = (H_t + \tau_t I)^{-1}, \qquad \tau_t = 10^{-10} + 10^{-8}\max\left(\frac{\mathrm{tr}(H_t)}{k}, 0\right). \tag{15}$$

The correction is applied through Cholesky solves, with an eigenvalue-floor fallback if factorization fails. The implemented reward is $R_t = \log[(\|r_t\|_2 + 10^{-8})/(\|r_{t+1}\|_2 + 10^{-8})]$, with no additional reward shaping in the default configuration.

Table 5: **Default training configuration.** Minibatches contain policy transitions; a smaller final minibatch is retained.

Setting	Value
Network and action parametrization	
Hidden width / message-passing rounds	32 / 2
Low-rank dimension	4
Initial Gaussian log standard deviation	-1 per action coordinate
Diagonal / basis increment bounds	± 0.5 / ± 0.25
Accumulated diagonal / basis bounds	± 5 / ± 2.5
PPO optimization	
Learning rate	3×10^{-4}, constant
Adam (β_1, β_2) / ϵ	$(0.9, 0.999)$ / 10^{-8}
Weight decay	0
PPO clipping parameter	0.2
Discount factor γ / GAE parameter λ	1 / 0.95
PPO epochs / maximum minibatch size	3 / 512
Value-loss coefficient	0.5
Entropy coefficient	0
Advantage normalization	Across the rollout batch
Maximum gradient ℓ_2 norm	0.5
Rollouts and budget	
Episodes per update	256
Training horizon T	25 solver steps
PPO updates per full training run	10,000
Richardson step size during training	$\alpha = 0.3$

Training procedure and variants. Each rollout batch draws new systems from the relevant problem distribution. Episodes run for the full horizon, with zero terminal bootstrap value. A full default run therefore contains 2.56×10^6 sampled episodes and 6.4×10^7 solver steps, excluding evaluation.

For static policies, the single decision specifies absolute diagonal parameters in $[-5, 5]$ and canonical basis entries in $[-2.5, 2.5]$. The episode contributes one PPO transition whose reward is the sum over its T solver steps. For policies with less frequent decisions, rewards are summed between successive decisions and GAE is applied at the decision level, retaining $\lambda = 0.95$. These variants retain the episode and solver-step budgets but use fewer policy transitions and optimizer steps per rollout batch.

The open-loop policy retains per-iteration preconditioner updates but removes both r_t and x_t from its observations. Its updates therefore depend on (A, b, t) without access to the evolving solver state. It otherwise follows the default training procedure.

Ablation-specific changes are described alongside the corresponding experiments in the main text and appendix.

A.3 Baseline Configuration and Tuning

All classical baselines use Richardson iteration with a preconditioner constructed once per system and held fixed throughout the solve. Table 6 specifies the implementations. ILU and AMG are constructed from the original matrix without an additional diagonal shift, including for the singular variable-coefficient diffusion systems described in Appendix C.1. Their construction parameters are fixed across problem families; only the Richardson step size is tuned.

Step-size selection. For each method and problem family, we select

$$\alpha \in \{0.025,\ 0.05,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.75,\ 1.0\} \tag{16}$$

to minimize the median relative residual after $T = 25$ iterations over a shared set of 200 tuning systems. Construction failures are counted separately and omitted from the tuning median. For each evaluated learned model, we tune the Richardson step size using the same grid and objective, keeping the policy weights fixed.

Table 6: **Baseline implementations and fixed settings.** The diagonal inverse actions include the numerical offsets used in the implementation, with $\epsilon = 10^{-8}$.

Baseline	Inverse action or configuration
Identity	$(1+\epsilon)r$.
Jacobi	$[P^{-1}r]_i = r_i/(\lvert A_{ii}\rvert + 2\epsilon)$.
ILU	SciPy `spilu` (SuperLU), with drop tolerance 10^{-4}, fill factor 4, COLAMD column ordering, and `basic,area` dropping rules. Application uses the factor object's solve routine.
AMG	PyAMG smoothed aggregation with one V-cycle per application, initialized at zero. Symmetric strength of connection (threshold 0), standard aggregation, and an initially constant near-nullspace candidate. Jacobi-smoothed prolongation; one symmetric block Gauss–Seidel sweep for both pre- and post-smoothing. At most 10 levels, coarse-size threshold 5, and a pseudoinverse coarse solver.

Retuning. The anisotropy sweep repeats tuning for each anisotropy ratio. At new problem sizes, baseline and learned-policy step sizes are retuned using the same grid and objective over 50 systems per size, without retraining the policies. Longer-solve experiments reuse the corresponding short-horizon step sizes rather than optimizing them for the target tolerance. The selected step size remains constant throughout each solve.

A.4 Evaluation Protocol

For each learned policy, we select the checkpoint with the lowest median final relative residual on 200 validation systems generated with sampling seed 54321. These systems are also used for step-size tuning, except that size-transfer tuning uses the smaller sample specified in Appendix A.3. At evaluation, we apply the selected policy deterministically using its mean action. All methods start from $x_0 = 0$ and solve the same 200 test systems, generated with sampling seed 12345. The timing experiments instead use 100 systems per size, shared across methods, with the protocol specified in Appendix A.5.

Fixed-horizon evaluations run for $T = 25$ iterations. For longer CLP solves, we use the defect-correction procedure of Section 4.4. Iteration counts accumulate across correction cycles, and the stopping criterion uses the residual relative to the original right-hand side, $\|b - Ax\|_2/\|b\|_2$. Runs that fail to reach the tolerance are counted as nonconverged. The evaluations in Figures 1 and 2 and Table 1 allow up to 35,000 iterations per method.

A.5 Compute Environment and Timing Protocol

Compute environment. Experiments used a workstation with two NVIDIA RTX A6000 GPUs with 48 GB of memory each, with each individual run using a single GPU. The software environment comprised Python 3.11, PyTorch 2.5, and CUDA 12.1, with SciPy 1.17 and PyAMG 5.3 providing the classical preconditioners. Policy networks use single precision, while matrix assembly and evaluation solver arithmetic use double precision.

Timing setup. The timing experiments in Section 5.2 evaluate magnetized diffusion at sizes up to $n = 9216$. All methods solve the same 100 systems at each size, one system at a time. CLP uses the policy trained at $n = 256$ with $T = 25$, reused without retraining through the defect-correction procedure of Section 4.4. Richardson step sizes are retuned at each size as described in Appendix A.3. Classical preconditioners are constructed on the CPU with eight threads; all iterations execute on the same GPU, with the operator and solver state retained on the device. The learned policy is compiled before measurement.

Measurement procedure. After two warm-up solves per method, setup and transfers are measured with wall-clock timers, and GPU operations with synchronized CUDA-event timings. Reported solver time includes setup and operations through the first tolerance crossing, including observation construction, policy inference, preconditioner updates, and application. For CLP, it also

includes residual recomputation and preconditioner reinitialization between defect-correction cycles. Training, compilation, system generation, host overhead outside the timed windows, and work after the crossing are excluded.

Time-to-tolerance reporting. The size comparison targets a relative residual of 10^{-8}; the tolerance sweep reports crossings at the displayed thresholds. Each method is allowed up to $35{,}000$ iterations. Systems that do not reach the target tolerance contribute $+\infty$ to iteration and time-to-tolerance summaries. Table 1 defines the cost categories and reports per-iteration costs using within-solve medians.

B Additional Experiments

B.1 Observation Ablation

We extend the feedback comparison in Section 5.3 by examining the contribution of individual observation channels. Each variant is retrained after removing one channel, retaining the operator graph, preconditioner parametrization, and training budget. For reference, Table 7 also includes the open-loop policy, which removes both r_t and x_t and updates the preconditioner using only (A, b, t).

Table 7: **Observation ablation on magnetized diffusion.** Checkmarks indicate retained observation channels. Δ denotes the increase in the reported score relative to the full observation.

	Observed					
Policy	r_t	x_t	b	t/T	$\log_{10}(\|r_T\|_2/\|r_0\|_2)$ ($\downarrow$)	Δ
CLP (Full observation)	✓	✓	✓	✓	$\mathbf{-0.72 \pm 0.13}$	–
Without t/T	✓	✓	✓	–	-0.70 ± 0.17	$+0.02$
Without b	✓	✓	–	✓	-0.66 ± 0.15	$+0.06$
Without x_t	✓	–	✓	✓	-0.56 ± 0.24	$+0.16$
Without r_t	–	✓	✓	✓	-0.44 ± 0.05	$+0.28$
Open-loop (without r_t and x_t)	–	–	✓	✓	-0.49 ± 0.04	$+0.23$

The full observation achieves the lowest reported final residual. Among the single-channel ablations, removing the residual causes the largest degradation, followed by removing the iterate. Removing the right-hand side or normalized iteration index has a smaller effect on the reported score. These comparisons support retaining both residual and iterate feedback in CLP, complementing the open-loop comparison in Section 5.3.

B.2 Decision Frequency

We examine how frequently the policy needs to update the preconditioner, training separate policies with decision intervals $k \in \{1, 3, 5, 7\}$. Here k denotes the number of Richardson iterations between policy decisions; the low-rank dimension remains fixed at 4. Between decisions, the preconditioner remains fixed. The point at $k = 25$ corresponds to the separately trained one-shot static policy. All variants retain the network architecture, rank, and solver-step training budget of the default configuration. Only the action bounds are adjusted across decision intervals, so that all variants can reach the same range of preconditioner parameters over the solve.

Figure 6 shows progressively weaker residual reduction as decisions become less frequent. Updating every iteration gives the best result, while larger intervals approach the performance of the static policy. This supports frequent within-solve updates under the tested training procedure. The comparison also changes policy computation and the number of PPO transitions and optimizer steps, as detailed in Appendix A.2. It therefore complements the observation ablation in Appendix B.1, which varies access to solver feedback while retaining per-iteration updates.

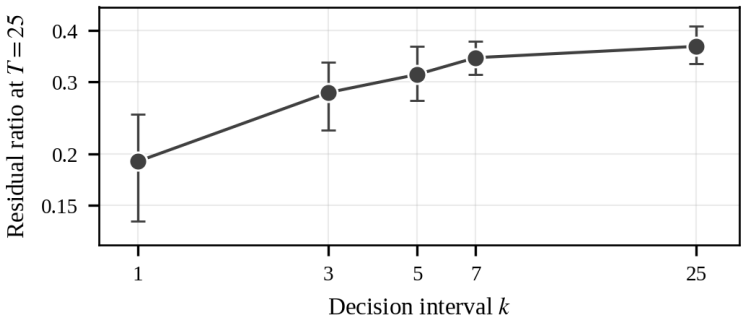

Figure 6: **Decision frequency on magnetized diffusion.** More frequent preconditioner updates yield lower final residuals. The decision interval k counts Richardson iterations between updates; $k = 25$ denotes the one-shot static policy. Error bars show instance IQRs.

B.3 Permutation Consistency

Equivariance. The ordering of unknowns is a storage convention and should not affect the solver's behavior. The graph policy respects this symmetry: relabelling the unknowns permutes the graph, node observations, and action outputs consistently. Writing $P_{t+1}^{-1}(A, r_t, x_t, b)$ for the updated preconditioner produced by the deterministic mean-action policy, we have

$$P_{t+1}^{-1}\big(\Pi A\Pi^\top,\, \Pi r_t,\, \Pi x_t,\, \Pi b\big) = \Pi\, P_{t+1}^{-1}(A, r_t, x_t, b)\, \Pi^\top \tag{17}$$

for any permutation matrix $\Pi \in \mathbb{R}^{n\times n}$. The dependence on the accumulated preconditioner state is implicit; its diagonal parameters and low-rank basis rows are permuted consistently.

This property follows from shared node-wise transformations and permutation-invariant aggregation over neighbors. Observation canonicalization and normalization commute with relabeling, while the Galerkin matrix is unchanged because $(\Pi U_{t+1})^\top(\Pi A\Pi^\top)(\Pi U_{t+1}) = U_{t+1}^\top A U_{t+1}$. The decoded diagonal and low-rank correction therefore transform as stated above. Thus, starting from $x_0 = 0$ and consistently permuted preconditioner states, Richardson iteration produces correspondingly permuted iterates and identical residual norms in exact arithmetic. The critic instead pools node representations into a scalar, making its output permutation invariant.

Empirical test. We test this consistency on magnetized diffusion, as discussed in Section 5.4. For each test system, we randomly relabel the unknowns, transforming (A, b) to $(\Pi A\Pi^\top, \Pi b)$, and evaluate the same trained policy on both representations. This preserves the underlying linear system and the Euclidean residual norm, while changing the ordering of the policy inputs.

Table 8: **Permutation consistency on magnetized diffusion.** Original and permuted scores are $\log_{10}(\|r_T\|_2/\|r_0\|_2)$. For each policy, we report the test instance with the largest absolute score change, $|\Delta|$, under relabelling.

Policy	Original	Permuted	$\lvert\Delta\rvert$
MLP	-0.399	$+2.378$	2.8
GNN (CLP)	-0.760	-0.760	0.0

Table 8 reports, separately for each policy, the test instance with the largest absolute change in the final log-relative residual. These are worst-case instance comparisons rather than aggregate performance summaries. The largest change for the graph policy is 7.6×10^{-5}, consistent with its permutation equivariance up to numerical differences. For the MLP, relabelling can instead change residual reduction into residual growth.

B.4 Parametrization and Rank

We specify the alternative weightings compared in Section 5.4, then examine the components and rank of the preconditioner. All variants retain the default graph backbone and training budget.

Alternative subspace weightings. At rank 4, we compare Galerkin weighting with three alternatives of the form $C_t = \mathrm{diag}(g_{t,1}, \ldots, g_{t,k})$. These provide fixed, learned, and analytic weights for the learned basis directions. Their nonnegative weights preserve the positive semidefiniteness of the low-rank correction and hence the positive definiteness of the full preconditioner.

Identity weighting fixes $g_{t,j} = 1$. It provides a reference in which the policy learns the basis without a separate weighting mechanism.

Learned weighting uses $g_{t,j} = \exp(\ell_{t,j})$, allowing the policy to adjust the weights through the same reinforcement learning objective as the basis. A linear head applied to the mean-pooled actor embeddings provides k additional action means for log-gain increments. The log-gains are initialized at zero, updated by increments bounded by 0.5, and clipped to $[-5, 5]$. This tests whether independently learned positive weights can provide an effective alternative to analytic Galerkin weighting.

Rayleigh-quotient weighting provides an analytic alternative based on the curvature of A along each basis direction:

$$\lambda_{t,j} = \frac{u_{t,j}^\top A u_{t,j}}{\max(\|u_{t,j}\|_2^2, \epsilon)}, \qquad g_{t,j} = \mathrm{clip}\left(\frac{1}{\max(\lambda_{t,j}, \epsilon)} - u_{t,j}^\top D_t^{-1} u_{t,j}, 0, \epsilon^{-1}\right),$$

where $u_{t,j}$ is a column of the low-rank basis, D_t^{-1} is the applied diagonal inverse, and $\epsilon = 10^{-8}$. This subtracts the diagonal contribution from an inverse Rayleigh-quotient target and retains a nonnegative correction. It tests operator-dependent weighting of individual directions, whereas Galerkin incorporates their coupling through the full regularized projected inverse specified in Appendix A.2. The weighting results are reported in Table 4.

Preconditioner components and rank. Table 9 examines the contribution of each learned component and the effect of the low-rank dimension. Removing the low-rank correction or freezing the diagonal weakens the reported residual reduction. The latter variant retains Jacobi scaling and learns only the low-rank basis, so the comparison concerns diagonal adaptation rather than the presence of a positive diagonal.

Table 9: **Preconditioner components and rank on magnetized diffusion.** The fixed-diagonal variant retains Jacobi scaling.

Parametrization	k	$\log_{10}(\|r_T\|_2/\|r_0\|_2)$ (↓)
Low-rank only	4	-0.57 ± 0.09
Diagonal only	0	-0.51 ± 0.04
Diagonal + low-rank	2	-0.67 ± 0.10
	4	-0.72 ± 0.13
	8	$\mathbf{-0.77 \pm 0.17}$
	16	-0.71 ± 0.17

Increasing the rank from 4 to 8 improves the reported score, but rank 16 provides no further gain. Rank 4 therefore remains a compact default: larger ranks increase basis storage, action dimension, and Galerkin-refresh cost, while the observed improvement is modest and nonmonotonic.

B.5 Network Capacity

We examine whether increasing policy capacity improves residual reduction, varying the hidden width and number of message-passing rounds while keeping the preconditioner rank and training budget fixed. Increasing width adds parameters, whereas additional message-passing rounds increase computation and receptive field without changing the parameter count, since weights are shared across rounds.

Table 10 shows no consistent improvement from either change. Width 64 does not outperform the default width 32, and additional message-passing rounds do not systematically improve the final residual. The smaller 16×1 network also approaches the default's performance with substantially fewer parameters. These results support the compact 32×2 architecture used throughout the paper,

Table 10: **Network capacity on magnetized diffusion.** Depth denotes message-passing rounds with shared parameters. The default configuration is 32×2.

Width $\times$ depth	Actor parameters	$\log_{10}(\|r_T\|_2/\|r_0\|_2)$ (↓)
16×1	2,053	-0.71 ± 0.14
16×2	2,053	-0.67 ± 0.14
32×1	7,685	-0.64 ± 0.08
32×2	7,685	$\mathbf{-0.72 \pm 0.13}$
32×3	7,685	-0.68 ± 0.10
64×2	29,701	-0.70 ± 0.13
64×3	29,701	-0.66 ± 0.11

while indicating that its performance does not depend on a large parameter count under the tested training budget.

C Supplementary Problem Families

We complement the magnetized diffusion experiments with three additional problem families: variable-coefficient diffusion, dense SPD matrices, and shifted random graph Laplacians. These cover isotropic diffusion with spatial heterogeneity, dense operators without sparse spatial structure, and sparse operators with irregular connectivity. The comparisons assess the scope of learned preconditioning beyond the main application. A separate policy is trained for each family using the configuration in Appendix A.2; these experiments do not test transfer of the magnetized diffusion policy.

C.1 Problem Generation

All three families use $n = 256$ unknowns. Variable-coefficient diffusion is discretized on a periodic 16×16 grid. The following descriptions specify the operator distributions and right-hand-side sampling used in these supplementary experiments.

Variable-coefficient diffusion. This family uses the isotropic diffusion tensor $\mathsf{D}(\boldsymbol{\xi}) = a(\boldsymbol{\xi})I$ and discretizes the operator $-\nabla \cdot (\mathsf{D}\nabla f)$ on an $n_x \times n_y$ periodic grid. It shares the isotropic tensor structure of magnetized diffusion when $\kappa_\parallel = \kappa_\perp$, but introduces spatially varying scalar diffusivity and uses a five-point discretization. The system matrix is the diffusion operator itself, without the backward-Euler identity term used in the magnetized family. Scalar coefficients are sampled independently at grid locations,

$$\log a_i \sim \mathrm{Uniform}(\log 0.1, \log 10), \qquad w_{ij} = \frac{2a_i a_j}{a_i + a_j}, \tag{18}$$

where w_{ij} is the harmonic mean for neighboring locations. The resulting conservative five-point operator acts on nodal values f_h as

$$(Af_h)_i = \sum_{j \in \mathcal{N}(i)} w_{ij}(f_i - f_j), \tag{19}$$

where $\mathcal{N}(i)$ contains the four periodic grid neighbors. The implementation uses unit grid spacing, omitting the common h^{-2} factor of a unit-square discretization. No diagonal shift is added.

All weights are positive and the grid is connected, so $\ker(A) = \mathrm{span}\{\mathbf{1}\}$: constant vectors form the entire nullspace. Consequently, this supplementary family extends the SPD setting of Section 3 to a singular positive semidefinite operator, which is positive definite on the subspace of mean-zero vectors, $\mathbf{1}^\perp$. Right-hand sides are projected onto this subspace according to Equation 23, ensuring that $Ax = b$ is consistent. Solutions are determined up to an additive constant.

Dense SPD matrices. We first sample a target condition number and a geometrically spaced spectrum,

$$\log \kappa \sim \mathrm{Uniform}(\log 10^2, \log 10^4), \qquad \lambda_i = \kappa^{(i-1)/(n-1)}, \quad i = 1, \ldots, n. \tag{20}$$

An orthogonal matrix Q is obtained by QR decomposition of a matrix with independent standard-normal entries. After randomly permuting the eigenvalues, we construct

$$\widetilde{A} = Q^\top \operatorname{diag}(\lambda_1, \ldots, \lambda_n) Q, \qquad A = \frac{\widetilde{A}}{\|\widetilde{A}\|_{\mathrm{F}} + 10^{-12}}. \tag{21}$$

The matrix is symmetrized numerically before normalization. This gives dense SPD systems with condition number κ and approximately unit Frobenius norm.

Random graph Laplacians. We construct an undirected, unweighted random graph with nominal density parameter $p = 0.06$. For reproducibility, the implemented construction uses a finite candidate sample: set $M = \lfloor pn(n-1)/2 \rfloor$, draw $2M$ ordered vertex pairs uniformly with replacement, retain pairs (i, j) satisfying $i < j$, and remove duplicates. If more than M distinct pairs remain, retain a uniformly sampled subset of size M. Thus, the sampler approximates the Erdős–Rényi graph Laplacian family rather than using independent Bernoulli draws for every possible edge.

Let W be the symmetric adjacency matrix, with unit weight on each retained edge. The system matrix is

$$A = \operatorname{diag}(W\mathbf{1}) - W + \mu I, \qquad \mu = 10^{-2}. \tag{22}$$

The positive shift guarantees that A is SPD, including when the sampled graph is disconnected; connectivity is not enforced.

Right-hand sides and sampling. For each episode, a new matrix is sampled from the selected family, and an independent vector $z \sim \mathcal{N}(0, I_n)$ is drawn. We use

$$b = \begin{cases} z - \dfrac{\mathbf{1}^\top z}{n}\mathbf{1}, & \text{variable-coefficient diffusion,} \\ z, & \text{dense SPD and random graph families.} \end{cases} \tag{23}$$

The projection ensures compatibility with the singular periodic diffusion operator.

C.2 Results Beyond Magnetized Diffusion

We compare CLP with classical preconditioners and a learned static policy on the three supplementary families. All methods use Richardson iteration and are evaluated at the trained horizon of $T = 25$, with step sizes tuned according to Appendix A.3. The learned static policy selects a single preconditioner at initialization and retains it throughout the solve, providing a reference for within-solve adaptation.

Table 11: **Preconditioning on supplementary problem families.** The best result per family is in bold, and the second best is underlined.

Method	Var. diffusion	Dense SPD	Random graph
Jacobi	-0.94 ± 0.09	-0.27 ± 1.09	-1.38 ± 0.29
ILU	-2.22 ± 0.21	$\mathbf{-13.80 \pm 0.40}$	-1.35 ± 0.30
AMG	$\mathbf{-9.35 \pm 0.68}$	-0.41 ± 0.36	$\mathbf{-14.11 \pm 0.28}$
Learned static	-0.98 ± 0.08	-0.29 ± 0.10	-2.10 ± 0.35
CLP (ours)	$\underline{-2.34 \pm 0.09}$	$\underline{-0.67 \pm 0.28}$	$\underline{-6.56 \pm 1.00}$

These families provide particularly favorable settings for established classical preconditioners. Variable-coefficient diffusion and shifted random graph Laplacians have a structure well suited to algebraic multigrid: relaxation reduces error that varies rapidly across strongly coupled unknowns, while coarse-space corrections address the remaining slowly varying components (Ruge & Stüben, 1987; Saad, 2003). For variable-coefficient diffusion, spatial heterogeneity preserves the scalar, isotropic diffusion structure. For the random graph family, the operator retains the Laplacian structure despite its irregular connectivity. The constant near-nullspace candidate used by our AMG baseline matches an exact null vector of the periodic diffusion operator and an eigenvector with

eigenvalue μ of the shifted graph Laplacian. These properties help explain AMG's strong performance: it achieves the lowest final residual on both families.

On the dense SPD family, ILU achieves near-machine-precision residuals and provides the strongest baseline. Here, the input matrix is already dense, so full triangular factors require storage of the same order as the matrix itself. With the fill allowance specified in Appendix A.3, the incomplete factorization can approach full LU, yielding a preconditioner close to a direct solve. In this dense-factor regime, conventional factorization requires $\mathcal{O}(n^3)$ setup work and $\mathcal{O}(n^2)$ storage, followed by $\mathcal{O}(n^2)$ work per preconditioner application. The fixed-horizon residual comparison reports the resulting accuracy but does not account for these setup and application costs.

CLP ranks second on all three families and improves on the learned static policy in each case (Table 11). The gaps to the strongest classical baselines delimit the scope of the present results: the tested CLP configuration does not match well-suited multigrid or factorization methods on these families. In particular, ILU on dense SPD systems and AMG on random graph Laplacians already achieve residuals near machine precision, leaving little meaningful room for improvement in the reported metric. CLP's consistent improvement over the learned static policy nevertheless supports the central contribution of this work: within-solve adaptation improves learned preconditioning across several operator structures, beyond the strongly anisotropic setting where CLP also outperforms the tested classical baselines.