Learning to Restart: Reinforcement Learning for Adaptive GMRES(m)

Authors. Ryan Divan (rd2700@princeton.edu),
Rishabh Mohapatra (rm5883@princeton.edu), Tyler Pellek (tp5523@princeton.edu)

Link to GitHub Repo. https://github.com/moogitus/GMRES_RL

Abstract

The GMRES(m) algorithm iteratively solves sparse linear systems, typically with a fixed restart parameter m. However, previous work has shown that variable choices of m can lead to stronger convergence behavior. We consider this problem in an online reinforcement learning (RL) framework and propose GMRES-RL, an online RL GMRES(m) algorithm. We find that compressed state-representations of the GMRES(m) residual achieve computational cost and accuracy that is competitive with fixed policies.

1 Introduction

One of the fundamental tools for problems in scientific computing is solving a linear system $Ax = b$, where $A \in \mathbb{C}^{n\times n}$ is a matrix, $b \in \mathbb{C}^n$ is a given vector, and $x \in \mathbb{C}^n$ is the unknown. These problems arise naturally in the solutions of discretized partial differential equations (PDEs), numerical optimization, and fluid dynamics, among other problems. Directly solving such systems provides robust solutions, but computational cost does not scale favorably. The direct solver of choice, an LU factorization followed by triangular solves, costs $\mathcal{O}(n^3)$ floating-point operations (FLOPs) and $\mathcal{O}(n^2)$ storage. As a result, when time and space are limited, practitioners look to alternative solvers.

The Generalized Minimal Residual (GMRES) method, proposed by Saad and Schultz, is an iterative solver that is particularly efficient for sparse systems [10]. It leverages the sparsity of matrices through solving the system via cheap matrix-vector multiplication, leading to low cost per iteration. GMRES guarantees convergence in n iterations, but is unique in that its cost scales quadratically in the number of iterations; k iterations of GMRES cost $\mathcal{O}(nk^2)$ FLOPs and $\mathcal{O}(nk)$ storage. Thus, in practice, GMRES(m), or restarted GMRES, is favored. GMRES(m) runs m GMRES iterations and uses the output vector as the initial guess for the next m iterations. This works well in resource-constrained contexts, limiting FLOPs to $\mathcal{O}(nm^2)$ and storage to $\mathcal{O}(nm)$, but may take more than n iterations to converge.

However, both forms of GMRES suffer a similar pitfall; it has been shown by Greenbaum [4] and Vecharynski & Langou [11] that GMRES and GMRES(m), respectively, can exhibit any possible convergence curve for a fixed matrix $A \in \mathbb{C}^{n\times n}$ by choosing an appropriately poor right-hand side $b \in \mathbb{C}^n$. In practice, this means GMRES may need the full n iterations to converge and GMRES(m) may never converge for $m < n$. Greenbaum also demonstrated that the spectrum of the matrix does not necessarily determine the behavior of GMRES. Furthermore, Embree showed that convergence of GMRES(m) is not monotone in m [2]. Altogether, these papers have shown that, given a constant GMRES restart rule, we can always find a system for which GMRES will converge arbitrarily slowly or even fail to do so. Practically and provably optimal choices of m have been the subject of decades of active research; while some rules perform well in many cases, the problem remains open.

We propose that reinforcement learning (RL) is a viable way to choose restart parameters for GMRES(m), learning an adaptive rule rather than relying on a fixed choice. While any fixed policy may suffer from an ill-conditioned system or a pathological example as Vecharynski & Langou might suggest, we examine GMRES-RL, an online reinforcement

learning approach. RL offers the possibility to learn a dynamic restart rule that can outperform deterministic and frozen stochastic rules on hard and pathological systems.

Empirically, our proposed GMRES-RL methodology matches or improves on AK-SLRL across the six original AK-SLRL benchmark matrices, reducing both Arnoldi work and wall-clock time while using a compressed controller state. On a 155-matrix SuiteSparse benchmark, GMRES-RL matches randGMRES in expected Arnoldi cost but improves reliability, raising convergence rate from 89.8% to 92.3% and reducing the difficult tail. A controlled convection-diffusion sweep further shows slower cost growth with condition number than both GMRES(20) and randGMRES. These results are limited to unpreconditioned GMRES with $m_{\max} = 20$, so preconditioned systems and larger restart ranges remain future work.

2 Related Work

The closest prior work studies GMRES(m) restart selection directly as an RL problem. Peairs and Chen first considered learning policies for varying the restart parameter in GMRES(m), showing that the restart length can be treated as a control action but reporting only marginal gains over existing methods [9]. Keramati and Hamdullahpur's AK-SLRL later made this framing more effective by using single-life RL: the agent learns online during the same solve, observes the current residual vector, and selects the next Krylov subspace dimension with a SAC controller [6]. We adopt AK-SLRL's core modeling insight that restart selection should be learned online and evaluated by Arnoldi work, since each restart cycle may contain a different number of iterations. Our contribution is to modify the parts that limit scalability: we replace the full residual-vector state with a fixed-dimensional convergence history, use DQN as $m \in \{1, \dots, m_{\max}\}$ is discrete, and use a work-penalized potential-shaped reward so the controller explicitly pays for larger Krylov subspaces.

3 Preliminaries

This section introduces the traditional GMRES(m) algorithm and notation. In particular, we distinguish a GMRES restart cycle from an individual Arnoldi step: the RL agent chooses the restart dimension m, but the computational cost of that choice is paid through the orthogonalization and projected least-squares solve. These preliminaries therefore explain both the solver being controlled and the work metric used later in the RL formulation.

The GMRES(m) algorithm solves the linear system $Ax = b$ where $A \in \mathbb{C}^{n\times n}$ is the matrix of coefficients, $b \in \mathbb{C}^n$ is the right-hand side vector, and $x \in \mathbb{C}^n$ is the unknown. When $m = n$, we call this the GMRES algorithm. We require an initial guess x_0 to set the residual $r_0 = b - Ax_0$. Then, GMRES(m) proceeds as follows at iteration $1 \le k \le n$:

1. Construct orthonormal basis V_{k+1} for *Krylov subspace* $\mathcal{K}_k(A, r_0) := \text{span}\{r_0, Ar_0, \cdots, A^{k-1}r_0\}$.
2. Reduce A to *upper Hessenberg* form, i.e. all entries below its first subdiagonal are zero, via Arnoldi iteration:
$$AV_k = V_{k+1}\hat{H}_k,$$
where $\hat{H}_k \in \mathbb{C}^{(k+1)\times k}$ is upper Hessenberg. This can be formed from $\hat{H}_{k-1}$.
3. Solve $y_k = \arg\min_{y\in\mathbb{C}^k} \|\hat{H}_k y - \|r_{k-1}\|_2 e_1\|_2$ and approximate solution $x_k = x_0 + V_k y_k$. Here, e_1 is the first standard basis vector, i.e. it is 1 in its first entry and 0 elsewhere.
4. Set $r_k = b - Ax_k$. If $\|r_k\|_2 < \epsilon_{\text{abs}}$ or $\|r_k\|_2/\|r_0\|_2 < \epsilon_{\text{rel}}$, for predetermined tolerance, then break and return x_k.
5. If $k = m \neq n$, then call GMRES(m) on the system $Ax = b$ with initial guess x_k.

Intuitively, GMRES(m) searches Krylov subspaces of increasing dimension for a best solution to the equation $Ax = b$; the solution is contained in the n-th order Krylov

subspace $\mathcal{K}_n(A, r_0)$. When the solution is found in a low-dimensional Krylov subspace, say $\mathcal{K}_k(A, r_0)$ for $k \leq m$, then GMRES(m) converges in k steps. However, if the solution is contained in $\mathcal{K}_k(A, r_0)$ for $k > m$, then GMRES(m) will take multiple cycles to converge, if at all, and often requires more than n total iterations across cycles.

The quadratic scaling of GMRES(m) in the number of iterations arises from Step 1, which uses the Gram-Schmidt algorithm (via a QR decomposition) to orthogonalize vectors to form a basis of $\mathcal{K}_k(A, b)$, which costs $\mathcal{O}(k^2 n)$ FLOPs. Storing these basis vectors costs $\mathcal{O}(kn)$ storage, and the upper Hessenberg matrix $\hat{H}_k$ costs $\mathcal{O}(k^2)$ in storage. Hence, it is desirable to bound k by some constant independent of n. While GMRES (with $m = n$) guarantees convergence in n iterations, linear scaling in n makes GMRES(m) more broadly used in practice.

4 Methodology

Our goal is to pose picking the GMRES restart parameter as an MDP and learn the optimal policy. In some cases, however, satisfying the Markov assumption may be expensive. In particular, using the full residual vector r as a state satisfies the Markov property (for an MDP defined on the system $Ax = b$), but this results in a computational burden that may be infeasible. Other natural choices for the state, such as $\|r\|_2$, suffer from the fact that the norm is not one-to-one, which depends on the residual vector itself and therefore previous residual norms. To this end, we introduce a d-th order Markov assumption for the system, providing the previous d residuals to balance computational complexity and robustness.

4.1 MDP and RL Cycle

One environment step for GMRES-RL corresponds to one GMRES restart cycle. At cycle k, the solver has iterate x_k and residual

$$r_k = b - Ax_k.$$

The agent observes a compact convergence history rather than the full residual vector:

$$s_k = (\|r_k\|_2,\, M_k,\, E_k)\,, \qquad M_k = (m_{k-d}, \ldots, m_{k-1}), \qquad E_k = (\|r_{k-d}\|_2, \ldots, \|r_{k-1}\|_2).$$

Here M_k and E_k store the previous d restart choices and residual norms. This is a practical d-th order Markov approximation: the full residual vector is more informative but high-dimensional, while the scalar residual norm alone is not one-to-one with future GMRES dynamics.

The action space is the finite set of restart dimensions

$$\mathcal{A} = \{1, 2, \ldots, m_{\max}\}.$$

After the agent selects m_k, the environment runs one GMRES(m_k) cycle, updates the solution to x_{k+1}, computes

$$r_{k+1} = b - Ax_{k+1},$$

and forms s_{k+1} by shifting the histories forward to include m_k and the new residual information. Thus, the transition kernel $\mathcal{T}$ is induced by the GMRES update, together with any randomness in the sampled matrix instance, initialization, or numerical environment; we do not estimate $\mathcal{T}$ explicitly. The learned policy $\pi(m \mid s)$ therefore selects restart parameters adaptively from recent convergence behavior rather than using a fixed value of m throughout the solve.

Although the original AK-SLRL framework uses Soft Actor-Critic (SAC) [6], our action space is inherently discrete. In the SAC formulation, the policy outputs a continuous action $a_t \in [0, 1]$, which is discretized as

$$m = \lfloor a_t(m_{\max} - 1) \rfloor + 1.$$

Thus, SAC solves a continuous relaxation of a finite restart-choice problem. We instead use DQN to learn $Q(s, m)$ directly over $m \in \{1, \ldots, m_{\max}\}$ and select restart dimensions by maximizing over these values. The reward function, defined next, balances residual progress against Krylov work. The GMRES-RL algorithm is given in Appendix 8.1.

4.2 Reward Function

The reward function connects GMRES convergence to the RL objective. Let $r_k = b - Ax_k$ denote the residual after restart cycle k. In AK-SLRL [6], the per-cycle reward $R_{\text{AK}}(r_{k-1}, m, r_k) = cte/\|r_k\|_2 + (\|r_{k-1}\|_2 - \|r_k\|_2)$, where cte is a constant, rewards both small residuals and one-cycle residual reduction. However, it does not explicitly price the cost of choosing a larger restart value m, which requires more Arnoldi steps, orthogonalization, and a larger projected least-squares solve.

We instead separate the computational objective from the residual-based learning signal. The base objective is the work penalty $R_{\text{work}}(r_{k-1}, m, r_k) = -\lambda_{\text{work}} m$. To provide dense feedback without changing the optimal policy of this base objective, we add potential-based shaping [8] with $\Phi_\tau(r) = -\log(\max(\|r\|_2, \tau)/\tau)$, so $\Phi_\tau(r) = 0$ once $\|r\|_2 \le \tau$. Thus,

$$R_{\text{PBRS}}(r_{k-1}, m, r_k) = -\lambda_{\text{work}} m + \gamma \Phi_\tau(r_k) - \Phi_\tau(r_{k-1}).$$

Equivalently, this is

$$-\lambda_{\text{work}} m + \log\left(\frac{\max(\|r_{k-1}\|_2, \tau)}{\tau}\right) - \gamma \log\left(\frac{\max(\|r_k\|_2, \tau)}{\tau}\right).$$

Our final reward adds a sparse bonus for first reaching tolerance: $R_{\text{final}} = R_{\text{PBRS}} + B\mathbf{1}\{\|r_k\|_2 < \tau \le \|r_{k-1}\|_2\}$. Thus, residual reduction supplies dense shaping feedback, while the underlying objective explicitly penalizes Arnoldi work. Bayesian optimization selected $(\lambda^*_{\text{work}}, \gamma^*, B^*) = (0.01, 0.925, 9)$, which we use for all GMRES-RL experiments; details of this search and the remaining hyperparameters are given in Table 5.

4.3 Reward Similarity via EPIC

We use the Equivalent-Policy Invariant Comparison (EPIC) distance [3] to compare reward functions directly, without training a separate policy for each. EPIC canonicalizes rewards before comparison, so it assigns zero distance to rewards differing only by positive rescaling or by a potential-based shaping term $\gamma\Phi(s') - \Phi(s)$. This makes it a natural diagnostic for our reward, which is constructed as a PBRS transformation of the work objective: any nonzero distance to R_{AK} must come from the underlying objective rather than the shaping.

We evaluate EPIC on two empirical coverage distributions D. The first, D_{conv}, is a controlled set generated from synthetic 1D convection-diffusion systems and provides a clean diagnostic on a single structured matrix family. The second, D_{HPO}, is generated from the 16 heterogeneous SuiteSparse matrices used for reward hyperparameter selection, testing whether the conclusion persists on the benchmark-aligned distribution that tuned the final reward parameters. The superscript (B) denotes inclusion of the sparse convergence bonus; transition collection, canonicalization, and bootstrap procedures are in Appendix 8.3.

Table 1: Pairwise EPIC distances on D_{conv} and D_{HPO}; values in $[0, 1]$, lower is closer after removing positive rescaling and PBRS. Our PBRS reward is policy-equivalent to its underlying work objective (zero distance), while R_{AK} and R_{PBRS} sit in distinct equivalence classes on both coverage sets, so the two rewards can favor different policies whenever larger m trades Arnoldi work for residual reduction.

Comparison	Coverage	EPIC distance	Interpretation
$R^{(0)}_{\text{PBRS}}$ vs. $R^{(0)}_{\text{work}}$	Convdiff	0.000	PBRS invariance check
$R^{(9)}_{\text{PBRS}}$ vs. $R^{(9)}_{\text{work}}$	HPO	0.000	PBRS invariance check
$R^{(9)}_{\text{AK}}$ vs. $R^{(9)}_{\text{PBRS}}$	Convdiff	0.400 $[0.353, 0.452]$	Different objectives
$R^{(9)}_{\text{AK}}$ vs. $R^{(9)}_{\text{PBRS}}$	HPO	0.494 $[0.404, 0.595]$	Difference persists

4.4 Metrics

We evaluate our algorithms with Arnoldi iterations and wall-clock time (s) until convergence. Within GMRES, one Arnoldi iteration is one step of the modified Gram–Schmidt

procedure that extends the orthonormal basis of the Krylov subspace by a single vector. Concretely, at iteration j we compute

$$h_{ij} = \langle Av_j, v_i \rangle, \quad i = 1, \ldots, j, \qquad v_{j+1} = \frac{1}{h_{j+1,j}} \left(Av_j - \sum_{i=1}^{j} h_{ij}\, v_i \right), \tag{1}$$

with $h_{j+1,j} = \left\| Av_j - \sum_{i=1}^{j} h_{ij}\, v_i \right\|_2$, producing the next basis vector v_{j+1} and column j of the upper Hessenberg matrix H_m.

Arnoldi iterations are our primary metric because the cost of this step is comparable across all choices of m encountered during learning. Optimizing over cycles to convergence, by contrast, would unfairly favor large m, since a single cycle with larger m performs strictly more work than a cycle with smaller m. Minimizing Arnoldi iterations therefore targets the dominant action-dependent computational cost of the SLRL GMRES algorithm without privileging any particular choice of m.

Wall-clock time serves as a secondary check, accounting for the action-invariant FLOPs and memory overhead introduced by the reinforcement learning machinery itself. All wall-clock measurements in this study were conducted on an M4 Max 16-core CPU.

For the convection-diffusion sweep we also report the spectral condition number $\kappa_2(A) = \sigma_{\max}(A)/\sigma_{\min}(A)$ of each system, computed exactly via dense SVD. GMRES's convergence rate is bounded above in terms of the condition number [10], so within a fixed matrix family $\kappa_2(A)$ serves as a single-scalar proxy for problem difficulty: larger values typically demand more Arnoldi iterations to reach a given residual tolerance.

Across the 155-matrix suite, we additionally report the rliable [1] interquartile mean (IQM) and probability of improvement (PoI) with stratified bootstrap 95% CIs, and McNemar's exact test [7] on the paired convergence indicator.

5 Results

We evaluate GMRES-RL along three experimental axes. First, we compare against AK-SLRL on its original six-matrix benchmark. Second, we test whether the gains persist statistically on a 155-matrix SuiteSparse benchmark against GMRES(20) and randGMRES. Third, we use a controlled convection-diffusion sweep to isolate how total Arnoldi work scales with condition number $\kappa_2(A)$.

Table 2: Performance on the six AK-SLRL benchmark matrices, mean±std over 5 seeds (GMRES(20) is deterministic, 1 run). Bold marks the best method per metric on matrices where every method converged on every seed. Markers: † residual $\geq 10\tau$ at the cap; ‡ partial convergence (1–4 of 5 seeds); ∘ all seeds within 10τ of tolerance but none strictly below. GMRES-RL improves on both AK-SLRL and GMRES(20) in Arnoldi count and wall-clock time on every matrix, with the gap over SAC widening with n as SAC's per-step overhead scales with the residual dimension.

	`1138_bus` (n=1,138)		`Finance256` (n=37,376)		`ct20stif` (n=52,329)	
Method	Arnoldi	Time (s)	Arnoldi	Time (s)	Arnoldi	Time (s)
GMRES(20)	200,000†	8.1	13,000	12.5	30,860	81.3
GMRES-RL (final)	**11,832** ±1,473	**1.6** ±0.3	**2,636** ±629	**2.3** ±0.6	**4,709** ±913	**11.1** ±1.9
AK-SLRL	17,868 ±3,215‡	107.0 ±35.6	3,143 ±995	20.3 ±9.6	17,156 ±4,440‡	586.3 ±500.9
	`Olesnik0` (n=88,263)		`ex19` (n=12,005)		`Crankseg_1` (n=52,804)	
Method	Arnoldi	Time (s)	Arnoldi	Time (s)	Arnoldi	Time (s)
GMRES(20)	200,000†	434.7	156,560	61.6	6,220	50.5
GMRES-RL (final)	42,284 ±4,693∘	72.0 ±11.0∘	**8,437** ±490	**3.2** ±0.2	**1,696** ±262	**13.6** ±2.2
AK-SLRL	109,209 ±45,177†	2,010.9 ±110.9	62,148 ±42,689‡	197.0 ±45.4	1,847 ±290	23.5 ±4.7

5.1 AK-SLRL Benchmark Comparison

We replicate the AK-SLRL evaluation on its six SuiteSparse matrices (`1138_bus`, `Finance256`, `ct20stif`, `olesnik0`, `ex19`, `crankseg_1`) [6]. AK-SLRL's full-residual SAC observation incurs $\mathcal{O}(n)$ controller-side memory per replay-buffer transition, which dominates the $\mathcal{O}(m_{\max} n)$ Krylov basis on the larger matrices and is infeasible at $n \gtrsim 10^5$. We replace the residual vector with a $2d$-dimensional rolling history of $d = 5$ log relative-residual norms and recent restart choices, giving a controller whose total overhead is $\mathcal{O}(1)$ in n.

GMRES-RL converges fully on five of six matrices and within an order of magnitude of tolerance on the sixth (○); AK-SLRL partially converges on three (‡) and fails on `olesnik0` (†). On the four matrices where every method strictly converged on every seed, GMRES-RL's geometric-mean speedup is 6.9× Arnoldi and 7.3× wall-clock over GMRES($m_{\max}$), and 2.4×/14.9× over AK-SLRL; on `ct20stif` AK-SLRL takes 586 s versus 11 s for GMRES-RL at comparable Arnoldi counts.

5.2 Statistical Comparison Against randGMRES on 155 SuiteSparse Matrices

Section 5.1 establishes parity with SAC but cannot support statistical scaling claims at $n = 6$. We extend to the Peairs-style 155-matrix SuiteSparse benchmark [9], spanning power networks, structural FEM, CFD, optimisation, electromagnetics, and graph problems with $n \in [10^3, 10^5]$ and nnz $\leq 10^6$ (Appendix 8.7). SAC is omitted on memory grounds; randGMRES (uniform sampling of $m \in \{5, 10, 15, 20\}$ per restart) is the strongest remaining stochastic baseline whose memory is bounded in n. All systems use $b = A\mathbf{1}$, $\tau = 10^{-6}$, and a 10^5-Arnoldi cap; stochastic methods run five seeds. Aggregates appear in Table 3 alongside IQM speedups and probability-of-improvement statistics over GMRES(20), computed with bootstrap CIs via rliable [1]; DQN hyperparameters are in Appendix 8.2.

5.2.1 Full Matrix Set Analysis

Table 3: Statistics on the 155-matrix benchmark, aggregated over all (matrix, seed) cells. *IQM* is the interquartile mean of per-cell Arnoldi speedup over GMRES(20) [1] (= 1.00 by construction). $P(>\text{GMRES(20)})$ is the probability a method's speedup strictly exceeds GMRES(20)'s on a matched draw (ties as 0.5). *Median seed-std* is the median across the 139 matrices on which both stochastic methods fully converged on every seed. *Conv. rate* is the fraction of cells reaching tolerance within the cap; † marks cap-hits. Bold marks the best in each column. GMRES-RL matches randGMRES on expected per-cell Arnoldi but strictly improves the p_{90}, the median seed-std, and the convergence rate.

	Arnoldi iterations					Wall-clock (s)		Median	Conv.
Method	Mean	Median	p90	IQM	$P(>\text{GMRES(20)})$	Mean	p90	seed-std	rate
GMRES(20)	39,001	10,480	100,000†	1.00	—	233.4	617.9	—	70.3%
randGMRES	15,809	**2,250**	100,000†	**3.11**	0.827	73.4	**138.2**	303	89.8%
GMRES-RL (final)	**15,488**	2,350	**65,205**	2.98	**0.863**	**69.1**	145.6	**232**	**92.3%**

The two stochastic methods are statistically indistinguishable on expected per-cell Arnoldi (rliable PoI = 0.525, 95% CI $[0.497, 0.553]$), and the +2.5 pp convergence-rate gap is significant under exact paired McNemar ($p = 3.8 \times 10^{-6}$, all 19 discordant cells favouring GMRES-RL; setup in Appendix 8.6).

5.2.2 Restricted Subset Analysis

We restrict to the 52 matrices with $n > 10^5$ rows or nnz $> 10^6$, the regime where restart choice actually moves the needle: each Arnoldi step costs $\mathcal{O}(\text{nnz} + mn)$ FLOPs and $V_m \in \mathbb{C}^{n \times m}$ consumes memory linear in n, so larger n or denser sparsity push compute and memory pressure into the range where adaptive restart is practically valuable.

Table 4: Same statistics as Table 3, restricted to the 52 matrices with $n > 10^5$ or nnz $> 10^6$ (52 cells for GMRES(20); 260 each for randGMRES and GMRES-RL). *Median seed-std* is across the 42 matrices on which both stochastic methods fully converged. Bold marks the best in each column. GMRES-RL's reliability advantage widens with hardness: the convergence-rate gap grows from +2.5 pp on the full benchmark to +5.7 pp here, and mean Arnoldi flips from a near-tie to a 2.4% GMRES-RL advantage.

	Arnoldi iterations					Wall-clock (s)		Median	Conv.
Method	Mean	Median	p90	IQM	$P(>\text{GMRES(20)})$	Mean	p90	seed-std	rate
GMRES(20)	59,330	74,250	100,000†	1.00	—	676.1	1,414.6	—	53.8%
randGMRES	27,321	**7,482**	100,005†	**4.21**	0.779	215.0	**679.5**	775	80.8%
GMRES-RL (final)	**26,656**	8,249	100,000†	3.97	**0.838**	**199.9**	698.4	**751**	**86.5%**

Per-cell cap-hits drop by 30% (50 → 35, vs the 24% reduction on the full benchmark) and full-matrix failures halve (10 → 5); randGMRES retains only a slight IQM-speedup edge on cells where both methods converge (4.21× vs 3.97× over GMRES(20)).

5.3 Convection-Diffusion Sweep over Condition Number

The benchmark in §5.2 is heterogeneous in size, sparsity, and conditioning, which prevents isolating how cost scales with any single property. To isolate conditioning we restrict to one class: 1D steady-state convection-diffusion operators discretised as tridiagonal upwind-stabilised differences of $-\varepsilon u'' + bu' = f$ on $[0,1]$ with Dirichlet boundary conditions, parameterized by ε and grid resolution n. The boundary-layer width $\sim \varepsilon/b$ controls $\kappa_2(A)$, so sweeping (ε, n) produces a one-parameter family whose condition number varies over orders of magnitude with the generating PDE held fixed. We generate 20 matrices with $n \in [500, 3500]$ and $\varepsilon \in [0.05, 0.25]$, yielding $\kappa_2(A) \in [3 \times 10^4, 2 \times 10^6]$. Right-hand sides are unit-normalised standard-normal. Tolerance, $m_{\max}$, and DQN hyperparameters match §5.2; cycle cap is 5,000 and $\kappa_2(A)$ is computed via dense SVD.

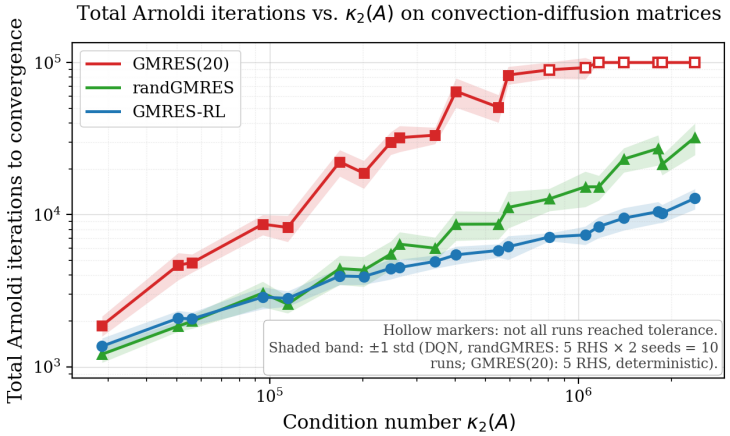

Figure 1: Mean total Arnoldi iterations to convergence on 20 1D convection-diffusion matrices vs $\kappa_2(A)$. Each point is one matrix; bands are ±1 std across 10 stochastic runs (5 RHS × 2 seeds) or 5 deterministic GMRES(20) runs; hollow markers denote matrices on which not every run converged. GMRES-RL's Arnoldi count grows the slowest with $\kappa_2(A)$ (log–log slopes 0.46 GMRES-RL, 0.73 randGMRES, 0.92 GMRES(20)), with the gap to randGMRES widening monotonically across the κ_2 range tested.

The reported GMRES(20) slope is a lower bound: the algorithm hits the cycle cap on the five hardest matrices ($\kappa_2 \gtrsim 10^6$), while GMRES-RL finishes below 13,000 Arnoldi iterations.

6 Discussion and Limitations

Three claims summarize the contribution. First, AK-SLRL's full-residual observation is principled, but unnecessary: the $2d$-dimensional convergence history preserves controller quality at $\mathcal{O}(1)$ overhead in n (Table 2), and the $\mathcal{O}(n)$ cost is not a constant factor. The SAC per-step overhead grows with n rather than amortizing and is infeasible on our hardware at $n \gtrsim 10^5$, so the SAC numbers in Table 2 are an upper bound on its best behavior. Second, GMRES-RL's advantage over randGMRES is in reliability and the worst-case tail rather than expected cost. The two are indistinguishable on per-cell Arnoldi (rliable PoI 0.525, 95% CI $[0.497, 0.553]$), but GMRES-RL's p_{90}, seed-std, and convergence rate are all favored, the McNemar test rejects interchangeability on convergence outcomes, and no matrix in the 155-matrix benchmark sees randGMRES converge while GMRES-RL fails. The reliability gap widens monotonically with hardness, on the $n > 10^5/\text{nnz} > 10^6$ subset (Table 4) and along the convdiff κ-sweep (slopes 0.46 vs 0.73 vs 0.92), so an adaptive controller is valuable on the larger-κ_2 tail where restart choice governs whether the solver converges at all. Third, AK-SLRL's residual-focused reward and our PBRS reward sit in different policy-equivalence classes (EPIC 0.400 $[0.353, 0.452]$ convdiff, 0.494 $[0.404, 0.595]$ HPO; Table 1, both intervals exclude zero), and the SAC-controlled ablation in Appendix 8.4 confirms it empirically: R_{final} achieves lower mean and standard deviation of total Arnoldi than R_{AK} on every level of the convdiff Easy–Extreme ladder (Table 6).

Several caveats bound these claims. We test unpreconditioned GMRES only and restrict to $m \in \{1, \ldots, 20\}$, so the larger-m regime is untouched. Five seeds per matrix suffices for the McNemar test but leaves per-matrix Arnoldi CIs wide, and the κ-sweep covers a single 1D PDE family at one discretization, so the 0.46 slope is in-family rather than universal. The 155-matrix Peairs benchmark (Appendix B) is filtered to square, positive-definite, $\geq 90\%$ symmetric SuiteSparse matrices, biasing evaluation away from the strongly non-symmetric and indefinite systems where GMRES is most distinctive over symmetric Krylov methods. Almost every experiment fixes $b = A\mathbf{1}$ (the κ-sweep uses unit-norm Gaussian RHS), so the pathological RHS constructions of [4] and [11] that motivate this work are not directly tested. The AK-SLRL paper does not specify several SAC parameters (learning rate, MLP architecture, batch size) and on the reward constant *cte*; we use stable-baselines3 defaults and $c_{\text{te}} = 1$, but a different setting could shift the SAC numbers we report.

7 Conclusion

We pose GMRES(m) restart selection as a single-life RL problem and learn it online with GMRES-RL, using DQN over a compressed convergence-history observation. GMRES-RL matches or beats SAC on every metric on the six AK-SLRL matrices with $\mathcal{O}(1)$ controller-side memory (Table 2), and on the 155-matrix Peairs-style benchmark matches randGMRES in expectation while strictly improving the tail, per-matrix seed-std, and convergence rate, with the gap widening on the harder subset (Table 4) and along the convection-diffusion κ-sweep (§5.3). An EPIC analysis and a SAC-controlled reward ablation show that the work-penalised PBRS reward sits in a different policy-equivalence class from AK-SLRL's (Table 1) and dominates it under a fixed SAC controller (Table 6). The AK-SLRL framing of single-life RL for GMRES restart selection is sufficiently motivated, but both its observation and reward can be simplified without losing controller quality.

7.1 Future Work

Our experiments isolate restart selection as our primary optimization for GMRES(m), but in practice GMRES benefits from preconditioner matrices [10]. A natural extension is testing whether the learned policy helps in that setting, or learning restart size and preconditioner jointly. Offline and online learning of GMRES preconditioners is an active problem [5].

Future work can also explore novel state representations. Our state representation offered the cheapest heuristic as GMRES already must compute residual norms. However, we conjecture observations such as residual angle, Ritz values, and Hessenberg matrix information may trade computational cost for stronger convergence behavior.

References

[1] Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. G. (2021). Deep reinforcement learning at the edge of the statistical precipice. *CoRR*, abs/2108.13264.

[2] Embree, M. (2003). The tortoise and the hare restart gmres. *SIAM Review*, 45(2):259–266.

[3] Gleave, A., Dennis, M., Legg, S., Russell, S., and Leike, J. (2021). Quantifying differences in reward functions. In *International Conference on Learning Representations*.

[4] Greenbaum, A., Pták, V., and Strakoš, Z. (1996). Any nonincreasing convergence curve is possible for gmres. *SIAM Journal on Matrix Analysis and Applications*, 17(3):465–469.

[5] Häusner, P., Juscafresa, A. N., and Sjölund, J. (2024). Learning incomplete factorization preconditioners for gmres.

[6] Keramati, H. and Hamdullahpur, F. (2025). Ak-slrl: Adaptive krylov subspace exploration using single-life reinforcement learning for sparse linear system. *arXiv preprint arXiv:2502.00227*.

[7] McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

[8] Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, pages 278–287, San Francisco, CA, USA. Morgan Kaufmann.

[9] Peairs, L. and Chen, T.-Y. (2011). Using reinforcement learning to vary the m in gmres(m). *Procedia Computer Science*, 4:2257–2266. Proceedings of the International Conference on Computational Science, ICCS 2011.

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

[11] Vecharynski, E. and Langou, J. (2009). Any decreasing cycle-convergence curve is possible for restarted gmres.

8 Appendix

8.1 GMRES Algorithm

Algorithm 1 GMRES-RL

Require: Matrix $A \in \mathbb{C}^{n \times n}$, right-hand side $b \in \mathbb{C}^n$, initial guess x_0, tolerance τ, maximum restart size $m_{\max}$

Compute initial residual $r_0 = b - Ax_0$
Set $x \leftarrow x_0$, $r \leftarrow r_0$, $k \leftarrow 0$
while $\|r\|_2 > \tau$ and maximum cycle count not reached **do**
    Construct RL state s_k from the current residual and recent history
    SLRL controller: select restart dimension

$$m_k \in \{1, \ldots, m_{\max}\}$$

    from the current policy/value estimate
    Store previous residual $r_{\text{prev}} \leftarrow r$
    GMRES cycle: build the Krylov subspace

$$\mathcal{K}_{m_k}(A, r) = \text{span}\{r, Ar, \ldots, A^{m_k - 1} r\}$$

    using m_k steps of Arnoldi iteration
    Obtain orthonormal basis V_{m_k} and Hessenberg matrix $\widehat{H}_{m_k}$
    Solve the projected least-squares problem

$$y_k = \arg\min_y \left\| \|r\|_2 e_1 - \widehat{H}_{m_k} y \right\|_2$$

    Update the solution

$$x \leftarrow x + V_{m_k} y_k$$

    Compute the new residual

$$r \leftarrow b - Ax$$

    Compute reward $R_k = R(r_{\text{prev}}, m_k, r)$ and update the SLRL agent using (s_k, m_k, R_k, s_{k+1})
    Set $k \leftarrow k + 1$
end while
return approximate solution x

8.2 Hyperparameters

We tuned only the reward coefficients $(\lambda_{\text{work}}, \gamma, B)$ in R_{final}; all other DQN and SAC baseline settings are fixed. The selected reward parameters are

$$\lambda^*_{\text{work}} = 0.01, \qquad \gamma^* = 0.925, \qquad B^* = 9,$$

and were used for all experiments. Our other hyperparameters can be seen in the table below.

Table 5: Hyperparameter search and fixed algorithm settings.

Category	Setting
Reward search	Optuna multivariate TPE; 15 random startup trials + 45 TPE-guided trials; tuned $(\lambda_{\text{work}}, \gamma, B)$ in R_{final}.
Search benchmark	Fixed suite of 16 SuiteSparse matrices with consistent right-hand side $b = A\mathbf{1}$.
Search space	$\lambda_{\text{work}} \in [10^{-4}, 10^{0}]$ log-uniform; $\gamma \in [0.80, 0.999]$ uniform; $B \in [0, 100]$ uniform.
Search objective	Mean total Arnoldi iterations to convergence across the 16-matrix suite; non-convergent solves receive penalty max_cycles $\cdot\, m_{\max}$, so any convergent setting strictly dominates any non-convergent one.
Selected reward values	$\lambda^*_{\text{work}} = 0.01$, $\gamma^* = 0.925$, $B^* = 9$.
DQN state	5-th order Markov assumption with depth $d = 5$; $s_k = (\|r_k\|_2, M_k, E_k)$ from §4.1, where M_k and E_k store the trailing five restart choices and log-relative-residual norms.
DQN budget / solver	Relative-residual tolerance $\tau = 10^{-6}$; cycle cap max_cycles = 10,000; maximum restart $m_{\max} = 20$.
DQN optimization	SB3-style single-life settings: Adam learning rate $\eta = 3 \times 10^{-3}$; replay buffer 10^4; batch size 32; hard target-network update every 100 gradient steps.
DQN exploration / network	ε-greedy exploration linearly annealed from 1.0 to 0.01 over the first 10% of the cycle budget; two-layer ReLU Q-network with hidden widths $[128, 128]$.
AK-SLRL SAC baseline	Paper-stated settings: $\gamma_{\text{SAC}} = 0.97$, replay buffer $\min(n/2, 20{,}000)$, automatic entropy tuning $\alpha =$ `auto` [6].
SAC defaults where silent	Stable-baselines3 defaults: learning rate 3×10^{-4}, batch size 256, Polyak coefficient $\tau_{\text{Polyak}} = 0.005$, target update every step, MLP widths $[256, 256]$.
AK-SLRL reward	R_{AK} uses $c_{\text{te}} = 1$ and no sparse convergence bonus [6].

8.3 EPIC Computation Details

EPIC compares reward functions on a fixed empirical coverage distribution D of transitions. In our setting, D is not generated by training a policy. Instead, we first collect GMRES restart transitions, then evaluate each reward function on this same transition set. This isolates reward-function similarity from policy optimization noise.

Each transition is generated by choosing a restart dimension m_i, running one true GMRES(m_i) cycle, and recording the reward-relevant tuple

$$z_i = (\rho_{i-1},\, m_i,\, \rho_i,\, \mathbf{1}\{\rho_i < \tau \le \rho_{i-1}\}), \qquad \rho_i = \frac{\|r_i\|_2}{\|b\|_2}.$$

Thus, the reward comparison uses relative residual norms, making the EPIC calculation scale-invariant across matrices. The final indicator records the first transition that crosses the convergence tolerance and is exactly where the sparse convergence bonus is applied.

We evaluate EPIC on two coverage distributions. The first, denoted D_{conv}, is a controlled coverage set generated from synthetic one-dimensional convection-diffusion systems. These systems provide a clean diagnostic because their residual dynamics come from a single structured matrix family. The second, denoted D_{HPO}, is generated from the 16 heterogeneous SuiteSparse matrices used in the reward hyperparameter sweep, with the same right-hand-side convention $b = A\mathbf{1}$. This second distribution tests whether the reward-similarity conclusions persist on the benchmark-aligned matrices used to tune the final reward parameters.

For both D_{conv} and D_{HPO}, transition collection uses a mixture of random and fixed-restart rollouts. In random rollouts, we sample

$$m_i \sim \text{Uniform}\{1, \ldots, m_{\max}\},$$

then run a real GMRES(m_i) cycle. These are not synthetic or fake transitions: only the restart choice is randomized, while the next residual is produced by the actual GMRES update. Random rollouts broaden action coverage, which is important because the rewards depend explicitly on m. Fixed $m = 20$ rollouts anchor the comparison to standard restarted GMRES.

Given a collected transition set

$$\mathcal{D} = \{z_i\}_{i=1}^{N},$$

we compute EPIC in four steps:

(1) Evaluate each raw reward $R(z_i)$ on every transition.
(2) Canonicalize each reward using empirical state/action marginals.
(3) Compute Pearson correlation between canonicalized reward vectors.
(4) Bootstrap over transitions to obtain confidence intervals.

For a reward R, EPIC constructs the canonicalized reward

$$C(R)(s, a, s') = R(s, a, s') + \mathbb{E}_{S,S' \sim D_S,\, A \sim D_A}\left[\gamma R(s', A, S') - R(s, A, S') - \gamma R(S, A, S')\right],$$

where D_S and D_A are the empirical state and action marginals induced by the same coverage set D. In our finite-sample implementation, these expectations are estimated by Monte Carlo averages over the recorded transition dataset. Concretely, for each evaluation transition (s_i, a_i, s_i'), we sample K empirical state-action-next-state triples $(\tilde{s}_j, \tilde{a}_j, \tilde{s}_j')$ from $\mathcal{D}$ and estimate

$$\widehat{C}(R)(s_i, a_i, s_i') = R(s_i, a_i, s_i') + \frac{1}{K}\sum_{j=1}^{K}\left[\gamma R(s_i', \tilde{a}_j, \tilde{s}_j') - R(s_i, \tilde{a}_j, \tilde{s}_j') - \gamma R(\tilde{s}_j, \tilde{a}_j, \tilde{s}_j')\right].$$

The last term is constant with respect to the evaluated transition (s_i, a_i, s_i'), so it does not affect Pearson correlation, but it is part of the theoretical canonicalization.

For two rewards R_A and R_B, we evaluate the canonicalized rewards on the same transition set,

$$\mathbf{c}_A = \left(\widehat{C}(R_A)(z_i)\right)_{i=1}^{N}, \qquad \mathbf{c}_B = \left(\widehat{C}(R_B)(z_i)\right)_{i=1}^{N},$$

and compute

$$\widehat{D}_{\text{EPIC}}(R_A, R_B) = \sqrt{\frac{1 - \text{corr}(\mathbf{c}_A, \mathbf{c}_B)}{2}}.$$

This distance lies in $[0, 1]$, where 0 means the rewards are equivalent up to positive rescaling and potential-based shaping on the sampled coverage distribution. Bootstrap confidence intervals are obtained by resampling $\mathcal{D}$ with replacement and recomputing the full EPIC distance.

For any reward R, the superscript B denotes the version with sparse convergence bonus:

$$R^{(B)}(r_{i-1}, m_i, r_i) = R(r_{i-1}, m_i, r_i) + B\mathbf{1}\{\rho_i < \tau \leq \rho_{i-1}\}.$$

In particular,

$$R^{(B)}_{\text{work}}(r_{i-1}, m_i, r_i) = -\lambda_{\text{work}} m_i + B\mathbf{1}\{\rho_i < \tau \leq \rho_{i-1}\},$$

and our potential-shaped reward is

$$R^{(B)}_{\text{PBRS}} = R^{(B)}_{\text{work}} + \gamma\Phi_\tau(r_i) - \Phi_\tau(r_{i-1}).$$

Therefore,

$$D_{\text{EPIC}}\left(R^{(B)}_{\text{PBRS}}, R^{(B)}_{\text{work}}\right) = 0.000000,$$

because the two rewards differ only by a potential-based shaping term. This is the formal sanity check; the reference rewards below are secondary baselines, not invariance checks.

On the heterogeneous HPO sweep matrices with $B = 9$, the full pairwise EPIC matrix is

	orig	pbrs	naive	work	linear	quad
orig	0.000	0.494	0.494	0.494	0.689	0.708
pbrs	0.494	0.000	0.000	0.000	0.481	0.524
naive	0.494	0.000	0.000	0.000	0.481	0.524
work	0.494	0.000	0.000	0.000	0.481	0.524
linear	0.689	0.481	0.481	0.481	0.000	0.102
quad	0.708	0.524	0.524	0.524	0.102	0.000

where "orig" denotes $R^{(9)}_{\text{AK}}$, "pbrs" denotes $R^{(9)}_{\text{PBRS}}$, "naive" denotes the unclamped log-shaped reward, "work" denotes $R^{(9)}_{\text{work}}$, and "linear" and "quad" denote the reference penalties $-m$ and $-m^2$, each with the same sparse convergence bonus.

The main HPO-sweep comparison is

$$D_{\text{EPIC}}\left(R^{(9)}_{\text{AK}}, R^{(9)}_{\text{PBRS}}\right) = 0.494381,$$

with bootstrap mean 0.492312, standard deviation 0.051503, and 95% percentile interval

$$[0.404063,\ 0.595369].$$

The corresponding convdiff result is

$$D_{\text{EPIC}}\left(R^{(9)}_{\text{AK}}, R^{(9)}_{\text{PBRS}}\right) = 0.399874, \qquad 95\%\ \text{CI} = [0.353443,\ 0.451524].$$

Thus, the conclusion is not an artifact of the controlled convection-diffusion coverage distribution: the original AK-SLRL reward and our PBRS reward remain separated on the heterogeneous benchmark-aligned transition distribution.

The reference penalties are included only for context. For example,

$$D_{\text{EPIC}}\left(R^{(9)}_{\text{work}}, R^{(9)}_{\text{ref-linear}}\right) = 0.481239$$

is nonzero because, with $B = 9$, the rewards

$$-\lambda_{\text{work}} m + 9\mathbf{1}\{\rho_i < \tau \le \rho_{i-1}\}$$

and

$$-m + 9\mathbf{1}\{\rho_i < \tau \le \rho_{i-1}\}$$

are not affine transformations of one another across both converging and non-converging transitions. If $B = 0$, they reduce to pure work penalties and are equivalent under positive rescaling.

8.4 Reward-Function Ablation under SAC

To isolate the reward function from the choice of RL algorithm, we hold the controller fixed at SAC and vary only the reward: the inverse-residual reward R_{AK} of Keramati and Hamdullahpur [6] and our potential-based reward R_{final} from §4.2. For each reward we run an independent Bayesian (Optuna TPE) hyperparameter search over its coefficients, (c_{te}, B) for R_{AK} and $(\lambda_{\text{work}}, \gamma, B)$ for R_{final}, on a five-matrix family of 1D convection-diffusion systems spanning $\kappa_2(A) \in [3.7 \times 10^4,\ 7.6 \times 10^5]$. Each evaluation uses 3 random right-hand sides and 3 independent seeds at relative-residual tolerance 10^{-6}. Table 6 reports total Arnoldi iterations to convergence at each reward's tuned optimum.

Table 6: SAC convergence under each reward. Mean±std total Arnoldi iterations over 3 RHS × 3 seeds. GMRES(20) is the deterministic fixed-restart baseline; bold marks the smaller mean between the two SAC variants. Notably, the SAC under R_{final} has a lower mean and standard deviation of Arnoldi steps until convergence for all 5 matrices

			GMRES(20)	SAC + R_{AK}		SAC + R_{final}	
Matrix	n	$\kappa_2(A)$	Arnoldi	Arnoldi	std	Arnoldi	std
Easy	1,000	3.7×10^4	9,947	16,319	23,354	**3,618**	478
Medium	1,000	1.2×10^5	29,867	41,829	41,205	**6,809**	760
Hard	1,000	1.9×10^5	42,313	22,430	5,059	**8,754**	1,077
Very Hard	1,500	4.3×10^5	110,053	30,978	23,013	**17,162**	2,639
Extreme	2,000	7.6×10^5	183,680	86,843	30,429	**27,563**	2,446

8.5 SuiteSparse Convergence Curves

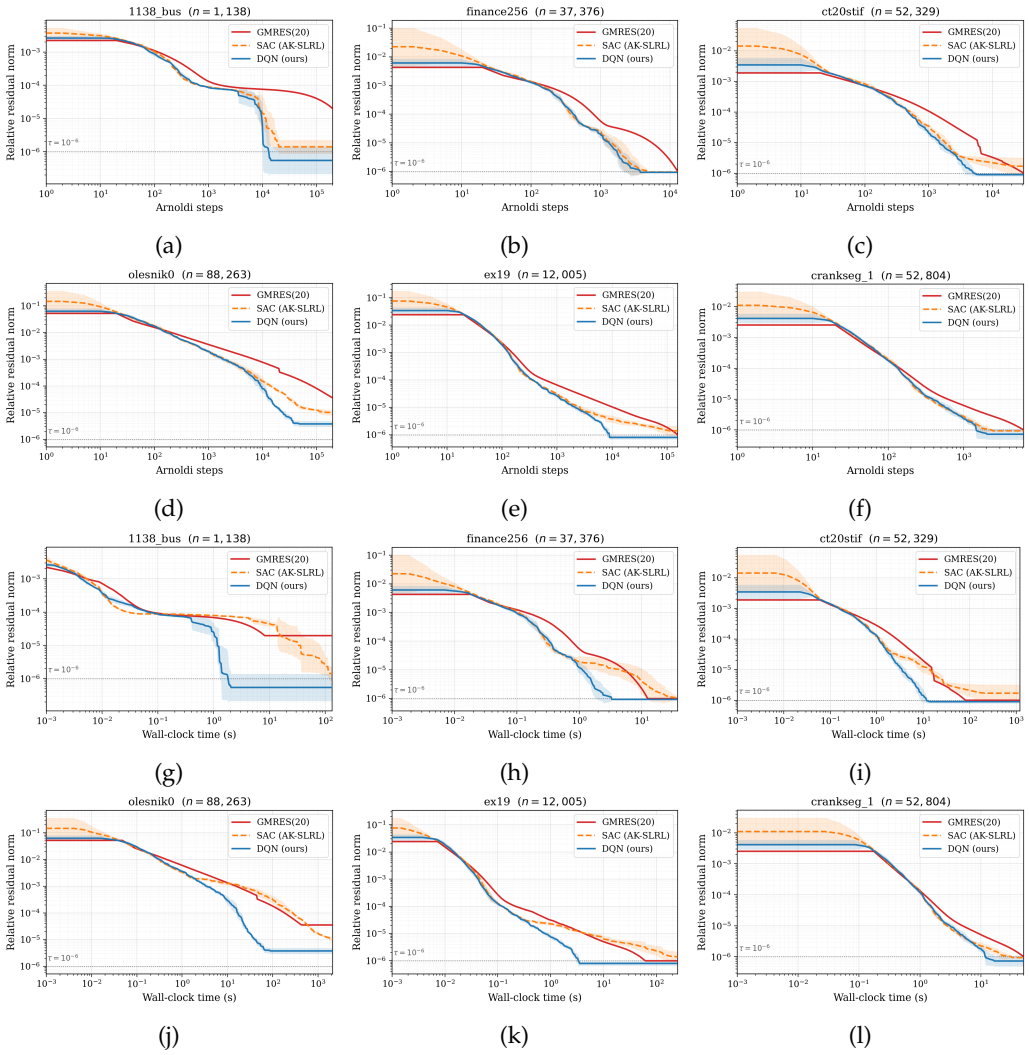

Figure 2: Convergence of GMRES(20), SAC (AK-SLRL), and DQN (ours) on the six AK-SLRL benchmark matrices: relative residual norm vs. Arnoldi steps and wall clock time. Solid line is the mean across seeds; shaded band is ±1 std. Horizontal dashed line marks the 10^{-6} relative-residual tolerance.

8.6 Convergence-Rate Hypothesis Test

This appendix supplies the formal setup, contingency table, and exact p-value behind the convergence-rate claim of §5.2. McNemar's test is a paired test for binary outcomes: it ignores cells where two methods agree and tests whether the two possible disagreement types occur equally often. The test is appropriate here because every $(\text{matrix}, \text{seed})$ cell evaluates GMRES-RL and randGMRES on the same linear system $Ax = b$, producing a matched binary outcome: whether the method reaches $\|r\|/\|b\| < 10^{-6}$ within the 10^5-Arnoldi cap. Let X_i and Y_i indicate convergence of GMRES-RL and randGMRES on cell i. The null hypothesis is

$$H_0 : \; P(X_i = 1, Y_i = 0) \;=\; P(X_i = 0, Y_i = 1),$$

i.e. that, conditional on the methods disagreeing, either method is equally likely to be the one that converges. Under H_0, the number of discordant cells favouring GMRES-RL follows $\text{Binomial}(n_{\text{disc}}, 0.5)$, and we report the exact two-sided binomial p-value. The observed contingency table across the 775 matched cells is given in Table 7.

Table 7: Per-cell convergence outcomes for the matched-pair McNemar test on the 775 $(\text{matrix}, \text{seed})$ cells where both GMRES-RL and randGMRES were evaluated [7]. Rows: GMRES-RL outcome. Columns: randGMRES outcome. Convergence is defined as $\|r\|/\|b\| < 10^{-6}$ within the 10^5-Arnoldi cap. Notably, there are no cells in which randGMRES converges while GMRES-RL fails.

	randGMRES converges	randGMRES fails	total
GMRES-RL converges	696	19	715
GMRES-RL fails	0	60	60
total	696	79	775

The off-diagonal cells are the only ones contributing to the test statistic. There are $n_{\text{disc}} = 19 + 0 = 19$ discordant cells, all of which favour GMRES-RL: not a single cell of the 775 saw randGMRES converge while GMRES-RL failed. Under H_0, observing a 19-vs-0 split (or any more extreme outcome) has probability

$$2 \binom{19}{0} 0.5^{19} \;=\; 3.81 \times 10^{-6},$$

the exact two-sided McNemar p-value, which rejects H_0 at any conventional significance level [7]. The marginal convergence-rate difference of $+2.45$ percentage points ($p_{\text{GMRES-RL}} - p_{\text{rand}} = 715/775 - 696/775$) thus reflects a statistically significant and qualitatively one-sided advantage for the learned controller.

8.7 SuiteSparse Benchmark Matrices

Table 8: SuiteSparse matrices used in our 155 matrix benchmark. Matrices are specified to be square, positive-definite, 90%+ numerical/pattern symmetric, have at most 1,000,000 rows and columns, and have at most 15,000,000 nonzero entries.

Problem Name	**Application Area**	**Size (n)**	**Nonzeros (nnz)**
1138_bus	Power network	1,138	4,054
2cubes_sphere	Electromagnetics	101,492	1,647,264
Andrews	Computer graphics/vision	60,000	760,154
BenElechi1	2D/3D	245,874	13,150,496
Chem97ZtZ	Statistical/mathematical	2,541	7,361
Dubcova1	2D/3D	16,129	253,009
Dubcova2	2D/3D	65,025	1,030,225

Problem Name	Application Area	Size (n)	Nonzeros (nnz)
Dubcova3	2D/3D	146,689	3,636,643
G2_circuit	Circuit simulation	150,102	726,674
G3_circuit	Circuit simulation	1,585,478	7,660,826
Kuu	Structural	7,102	340,200
LF10000	Model reduction	19,998	99,982
LFAT5000	Model reduction	19,994	79,966
Muu	Structural	7,102	170,134
Pres_Poisson	Computational fluid dynamics	14,822	715,804
Spielman_k100	Undirected weighted graph	338,402	1,025,404
Trefethen_2000	Combinatorial	2,000	41,906
Trefethen_20000	Combinatorial	20,000	554,466
Trefethen_20000b	Combinatorial	19,999	554,435
af01	Acoustics	8,205	125,567
apache1	Structural	80,800	542,184
apache2	Structural	715,176	4,817,870
bcsstk08	Structural	1,074	12,960
bcsstk09	Structural	1,083	18,437
bcsstk10	Structural	1,086	22,070
bcsstk11	Structural	1,473	34,241
bcsstk12	Duplicate structural	1,473	34,241
bcsstk13	Computational fluid dynamics	2,003	83,883
bcsstk14	Structural	1,806	63,454
bcsstk15	Structural	3,948	117,816
bcsstk16	Structural	4,884	290,378
bcsstk17	Structural	10,974	428,650
bcsstk18	Structural	11,948	149,090
bcsstk21	Structural	3,600	26,600
bcsstk23	Structural	3,134	45,178
bcsstk24	Structural	3,562	159,910
bcsstk25	Structural	15,439	252,241
bcsstk26	Structural	1,922	30,336
bcsstk27	Structural	1,224	56,126
bcsstk28	Structural	4,410	219,024
bcsstk36	Structural	23,052	1,143,140
bcsstk38	Structural	8,032	355,460
bcsstm08	Structural	1,074	1,074
bcsstm09	Structural	1,083	1,083
bcsstm11	Structural	1,473	1,473
bcsstm12	Structural	1,473	19,659
bcsstm21	Structural	3,600	3,600
bcsstm23	Structural	3,134	3,134
bcsstm24	Structural	3,562	3,562
bcsstm25	Structural	15,439	15,439
bcsstm26	Structural	1,922	1,922
bcsstm39	Structural	46,772	46,772
bibd_81_2	Combinatorial	3,240	3,240
bloweybq	Materials	10,001	49,999
bmw7st_1	Structural	141,347	7,318,399
bmwcra_1	Structural	148,770	10,641,602
bodyy4	Structural	17,546	121,550
bodyy5	Structural	18,589	128,853
bodyy6	Structural	19,366	134,208
boneS01	Model reduction	127,224	5,516,602
bundle1	Computer graphics/vision	10,581	770,811
cant	2D/3D	62,451	4,007,383
cbuckle	Structural	13,681	676,515
cfd1	Computational fluid dynamics	70,656	1,825,580

Problem Name	**Application Area**	**Size (n)**	**Nonzeros (nnz)**
cfd2	Computational fluid dynamics	123,440	3,085,406
consph	2D/3D	83,334	6,010,480
crankseg_1	Structural	52,804	10,614,210
crankseg_2	Structural	63,838	14,148,858
crystm01	Materials	4,875	105,339
crystm02	Materials	13,965	322,905
crystm03	Materials	24,696	583,770
ct20stif	Structural	52,329	2,600,295
cvxbqp1	Optimization	50,000	349,968
denormal	Counter-example	89,400	1,156,224
ex10	Computational fluid dynamics	2,410	54,840
ex10hs	Computational fluid dynamics	2,548	57,308
ex13	Computational fluid dynamics	2,568	75,628
ex15	Computational fluid dynamics	6,867	98,671
ex3	Computational fluid dynamics	1,821	52,685
ex33	Computational fluid dynamics	1,733	22,189
ex9	Computational fluid dynamics	3,363	99,471
finan512	Economic	74,752	596,992
fv1	2D/3D	9,604	85,264
fv2	2D/3D	9,801	87,025
fv3	2D/3D	9,801	87,025
gridgena	Optimization	48,962	512,084