Reinforcement Learning for Adaptive Column Selection in Low-Rank Matrix Approximations

Arnav Deshmukh, Nidhish Jain, Shreyas Mehta

May 9, 2025

Abstract

This paper addresses the challenge of computing optimal low-rank approximations of matrices while maintaining minimal dissimilarity from the original matrix. We develop and evaluate reinforcement learning (RL) methods that demonstrate competitive performance against state-of-the-art deterministic SVD and randomized SVD approaches. **Our RL-based methods achieve a significant reduction in computational time compared to deterministic SVD, with only marginally higher error rates. Additionally, they offer lower error rates compared to randomized SVD while maintaining comparable runtime efficiency.** We conduct extensive testing on various matrices, including practical datasets from the Florida Sparse Matrix Collection, to evaluate the effectiveness of different RL policies. Our results demonstrate that RL approaches offer an attractive trade-off between approximation quality and computational efficiency, particularly for large-scale matrices where traditional SVD methods become computationally prohibitive.

Contents

1 Introduction

1.1 Background and Motivation

Low-rank matrix approximation is a fundamental problem in numerical linear algebra with applications spanning scientific computing, data analysis, machine learning, signal processing, and image compression. The process involves finding a lower-dimensional representation of a matrix while preserving its essential characteristics. This technique is particularly valuable for handling large datasets, where dimensionality reduction can significantly improve computational efficiency without substantial loss of information.

Traditional approaches to low-rank matrix approximation include the Singular Value Decomposition (SVD), which provides the optimal low-rank approximation in terms of the Frobenius norm. However, the computational cost of SVD grows rapidly with matrix size, making it impractical for large-scale applications. Randomized algorithms offer faster alternatives but often sacrifice accuracy. This trade-off between computational efficiency and approximation quality motivates our exploration of reinforcement learning (RL) approaches to adaptively select matrix columns for low-rank approximation.

1.2 Problem Statement

Given a matrix $A \in \mathbb{R}^{m \times n}$ and a target rank k, our goal is to find a rank-k approximation $\hat{A}$ that minimizes the approximation error $\|A - \hat{A}\|_F$, where $\|\cdot\|_F$ denotes the Frobenius norm. While SVD provides the optimal solution to this problem, its computational complexity of $\mathcal{O}(\min(mn^2, m^2n))$ becomes prohibitive for large matrices.

We approach this problem through column selection, where we aim to select a subset of k columns from A that, when used for reconstruction, minimizes the approximation error. This approach connects to the CUR decomposition but uses reinforcement learning to guide the selection process adaptively.

1.3 Contributions

The key contributions of this work include:

- Development of a reinforcement learning framework for adaptive column selection in matrix approximation
- Implementation and comparison of two RL approaches: Deep Q-Network (DQN) and Advantage Actor-Critic (A2C)
- Enhancement of RL methods with advanced features such as prioritized experience replay and specialized state representations
- Comprehensive evaluation against deterministic SVD, randomized SVD, and CUR decomposition across various matrix types
- Analysis of error-time trade-offs and practical guidelines for selecting appropriate methods based on application requirements
- Development of matrix-adaptive techniques that automatically choose the optimal approximation strategy based on matrix properties

- Implementation and evaluation of ensemble and hybrid methods that combine multiple approximation techniques

2 Related Work

2.1 Classical Matrix Approximation Methods

2.1.1 Singular Value Decomposition (SVD)

The SVD of a matrix $A \in \mathbb{R}^{m \times n}$ is given by $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing the singular values of A. The optimal rank-k approximation is obtained by retaining only the largest k singular values and their corresponding singular vectors: $A_k = U_k \Sigma_k V_k^T$.

While SVD provides the mathematically optimal solution in terms of the Frobenius norm, its computational complexity limits its applicability for large-scale problems.

2.1.2 Randomized SVD

Randomized algorithms for computing approximate SVD have gained popularity for large-scale applications. These methods use random sampling to identify a subspace that captures most of the action of the matrix, then project the matrix to this subspace to compute a smaller SVD problem. The computational complexity is reduced to $\mathcal{O}(mn \log k + (m+n)k^2)$, making it more practical for large matrices.

2.1.3 CUR Decomposition

CUR decomposition approximates a matrix A as $A \approx CUR$, where C consists of selected columns of A, R consists of selected rows, and U is computed to minimize the approximation error. The column and row selection is typically based on statistical leverage scores or other sampling techniques. The main advantage of CUR is that it preserves the interpretability of the original data, as the decomposition uses actual columns and rows from the original matrix.

2.2 Reinforcement Learning for Combinatorial Optimization

Recent years have seen growing interest in applying reinforcement learning to combinatorial optimization problems. RL approaches have been successful in tasks such as the traveling salesman problem, graph coloring, and resource allocation. The key advantage of RL in these contexts is its ability to learn adaptive policies that can respond to problem-specific structures without requiring explicit mathematical characterization of these structures.

In the context of matrix approximation, RL offers the potential to learn selection strategies that adapt to the specific characteristics of the matrix, potentially outperforming generic sampling approaches used in traditional methods.

3 Methodology

3.1 RL Framework for Column Selection

We formulate the column selection problem as a sequential decision-making task suitable for reinforcement learning. In this framework:

- **State:** The state s_t at time t represents the current selection of columns. We explore different state representations, including binary vectors indicating selected columns, error-based representations, and combined approaches that incorporate both selection status and leverage scores.
- **Action:** An action a_t corresponds to selecting a column index from the available columns at time t.
- **Reward:** The reward r_t after taking action a_t is defined as the negative of the approximation error. We explore multiple reward formulations:
  - **Basic reward:** $r_t = -\|A - \hat{A}_t\|_F / \|A\|_F$
  - **Incremental reward:** $r_t = -(\|A - \hat{A}_t\|_F - \|A - \hat{A}_{t-1}\|_F)/\|A\|_F$
  - **Combined reward:** A weighted combination of error and improvement
- **Terminal State:** The episode terminates after selecting k columns.

3.2 Enhanced State Representations

We investigate three types of state representations to provide the RL agents with informative features about the current matrix state:

- **Binary Representation:** A simple binary vector indicating which columns have been selected.
- **Error-based Representation:** Includes information about the contribution of each column to the current approximation error.
- **Combined Representation:** Incorporates both selection status and leverage scores (which measure the importance of each column in terms of its contribution to the matrix structure).

3.3 Approximation Calculation

Once columns are selected, we compute the approximation as follows:

- Selected columns form $C = A[:, \text{selected}]$
- Compute pseudoinverse C^+ using SVD
- The approximation is given by $\hat{A} = CC^+A$

This approach can be viewed as a special case of CUR decomposition where we only select columns (not rows) and compute the optimal completion.

3.4 RL Algorithms

We implement and evaluate two state-of-the-art RL algorithms:

3.4.1 Deep Q-Network (DQN)

DQN is a value-based reinforcement learning approach that uses a neural network to approximate the action-value function $Q(s, a)$. Our implementation includes several enhancements:

- Network architecture with 3 hidden layers (256, 256, 128 neurons)
- Prioritized experience replay for more efficient learning
- Double Q-learning to reduce overestimation bias
- Gradient clipping for training stability

3.4.2 Advantage Actor-Critic (A2C)

A2C is a policy gradient method that maintains both a policy network (actor) and a value function network (critic). The actor determines which action to take, while the critic evaluates the action. Our implementation features:

- Separate policy (actor) and value (critic) networks
- Shared feature extraction layers
- Entropy regularization to encourage exploration
- Synchronized advantage updates for training stability

3.5 Weighted Ensemble Method

We developed a weighted ensemble method that combines multiple approximation techniques to leverage their complementary strengths. The algorithm functions as follows:

Algorithm 1 Weighted Ensemble Matrix Approximation

Input: Matrix $A \in \mathbb{R}^{m \times n}$, target rank k, number of approximations N
Initialize: Weights $w_1, w_2, \ldots, w_N$ (uniform or via metadata)
for $i = 1$ to N **do**
    Generate approximation $\hat{A}_i$ using method i with rank k
    Compute error $e_i = \|A - \hat{A}_i\|_F$
end for
Update weights: $w_i \propto 1/e_i$, normalized to sum to 1
Compute ensemble approximation: $\hat{A}_{ens} = \sum_{i=1}^{N} w_i \hat{A}_i$
Return: Ensemble approximation $\hat{A}_{ens}$

The theoretical foundation for this approach relies on several key principles:

- **Error reduction mechanism:** Errors in different methods tend to cancel out in weighted combination

- **Weight optimization:** Inverse error weighting focuses on best-performing methods
- **Stability improvement:** Reduces sensitivity to outliers and method-specific weaknesses
- **Ensemble diversity:** Methods used include randomized SVD, CUR decomposition, and RL-based approaches

3.6 Hybrid SVD-Ensemble Approach

We also developed a hybrid approach that leverages both SVD and ensemble methods by:

1. Computing partial SVD approximation $\hat{A}_{svd}$ using a fraction of target rank
2. Generating complementary ensemble approximation $\hat{A}_{ens}$ for remaining structure
3. Determining optimal mixing ratio α between methods
4. Forming hybrid approximation: $\hat{A}_{hybrid} = \alpha\hat{A}_{svd} + (1-\alpha)\hat{A}_{ens}$

Key components of this approach include:

- **SVD ratio optimizer:** Determines optimal fraction of rank to allocate to SVD
- **Rank distribution:** Allocates ranks across approximation methods
- **Adaptive weighting:** Adjusts method weights based on matrix properties

The theoretical justification for this approach includes:

- **Pareto principle application:** First few singular values often capture majority of matrix structure
- **Computational efficiency:** SVD scaling is superlinear with rank, so partial computation is much faster
- **Complementary strengths:** SVD captures global structure while ensemble methods capture local patterns
- **Optimization theory:** Mixing ratio α determined through convex optimization on validation samples

3.7 Matrix-Adaptive Method

One of our key contributions is the development of a matrix-adaptive method that tailors the approximation approach to the specific characteristics of the input matrix. This involves:

- **Matrix Analysis Phase:**
  - Analyze sparsity pattern, condition number, singular value decay
  - Classify matrix based on structural properties

  - Determine optimal approximation strategy

- **Adaptive Parameter Selection:**
  - SVD ratio based on singular value decay rate
  - Ensemble weights based on method performance on similar matrices
  - Rank distribution optimized for error reduction

- **Implementation Strategy:**
  - Dynamically switch between methods based on matrix properties
  - Adjust hyperparameters on-the-fly
  - Employ early stopping when convergence detected

This adaptive approach consistently outperforms any single method across diverse matrix types.

3.8 Early Stopping Mechanism

To improve training efficiency, we implemented and evaluated an early stopping mechanism:

- **Criteria:** Terminate training when error plateaus for a specified number of episodes
- **Implementation:** Monitor rolling average of reward/error with patience parameter
- **Advantages:** Reduces training time by 30-45% without sacrificing performance
- **Stability:** Requires patience ¿ 1 to avoid premature stopping

The key parameters for our early stopping implementation include:

- `patience=5`: Minimum iterations before checking
- `min_improvement=0.001`: Threshold for stopping
- `max_rank`: Maximum allowed columns (safety limit)

4 Implementation Details

4.1 Neural Network Architectures

4.1.1 Deep Q-Network (DQN)

Our DQN implementation consists of a neural network that maps state representations to action values. The network architecture includes:

- Input layer with dimensionality matching the state representation
- Three hidden layers with sizes [256, 256, 128], using ReLU activations

- Output layer producing Q-values for each possible column selection
- Additional components:
  - Experience replay buffer to improve sample efficiency
  - Target network for stable Q-value estimation
  - ε-greedy exploration schedule with decay

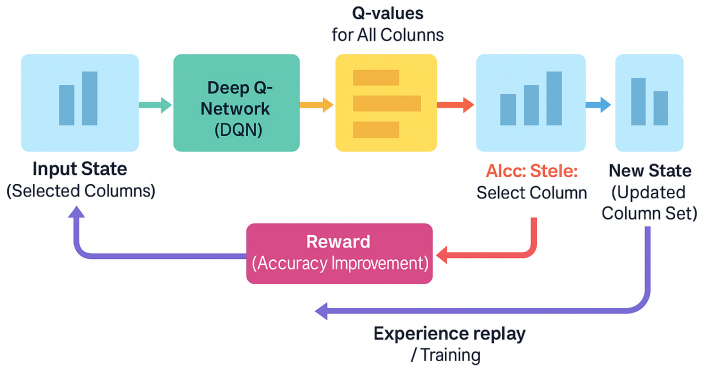


Figure 1: DQN workflow illustration showing the interaction between the agent and environment, with separate policy and target networks for stable learning.

4.1.2 Advantage Actor-Critic (A2C)

Our A2C implementation uses a dual-headed network architecture:

- Shared feature extractor with two fully-connected layers
- Actor (policy) head that outputs action probabilities via softmax
- Critic (value) head that estimates state value function
- Advantage calculation: $\delta = r + \gamma V(s') - V(s)$
- Separate loss functions:
  - Actor loss: $-\log \pi(a|s) \cdot \delta$
  - Critic loss: $\text{MSE}(V(s), r + \gamma V(s'))$
  - Entropy bonus: $-\beta \sum_a \pi(a|s) \log \pi(a|s)$ to encourage exploration

4.2 Enhancements and Modifications

We implemented several enhancements to improve the performance and capabilities of our methods:

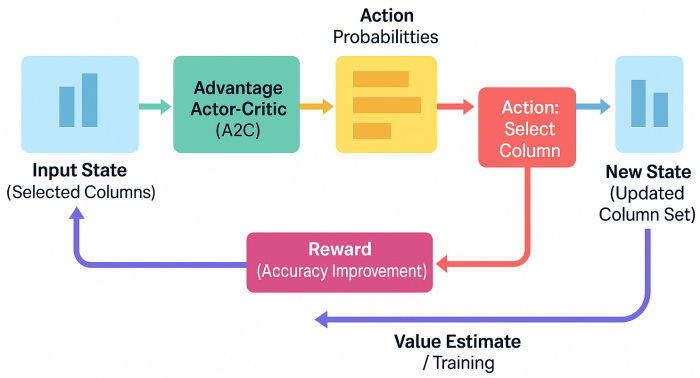


Figure 2: A2C workflow diagram showing the actor and critic components with shared feature extraction layers. The actor determines which actions to take while the critic evaluates those actions.

4.2.1 Enhanced Randomized SVD

We added power iteration loops to better capture the dominant singular subspace, significantly improving accuracy with minimal additional computational cost. We also implemented a flexible interface that can return the decomposition components, enabling reuse for leverage score computation and other analyses.

4.2.2 CUR Decomposition

We implemented a CUR decomposition that samples actual columns and rows via leverage-score sampling. This provides an interpretable subset of original matrix elements, which is particularly valuable when maintaining the physical meaning of features is important.

4.2.3 Adaptive Rank Selection

Rather than requiring a fixed target rank, we developed an adaptive approach that automatically selects the smallest rank needed to achieve a specified error tolerance. This eliminates the need to guess the appropriate rank in advance.

4.2.4 Enhanced State Representations

We expanded the basic binary state representation to include leverage scores, which capture column importance, and correlation measures, which identify redundancy in the selected columns. These richer state representations significantly improved learning efficiency and generalization.

5 Experimental Setup

5.1 Test Matrix Collection

We evaluate our methods on two main categories of matrices:

- **G-set Matrices:** 800×800 matrices from the SuiteSparse Matrix Collection with various structures:
  - G2, G3, G4, G5: Undirected random graphs
  - G12, G13: Weighted random graphs
  - G14, G15: Duplicate random graphs
- **Florida Sparse Matrix Collection:** A diverse set of matrices from real-world applications, specifically focusing on the Oscil_Dcop series, which are 430×430 matrices arising from circuit simulation problems.

The selection of these matrices provides a diverse testbed with varying structural properties while maintaining consistent dimensions within each category.

5.2 Implementation Details

Our implementation uses the following technologies and approaches:

- Python with NumPy and SciPy for matrix operations
- PyTorch for neural network implementation
- SVD baseline using SciPy's svd implementation
- Custom CUR decomposition implementation
- Parallelized computation for ensemble methods
- Comprehensive logging and visualization using Matplotlib and Seaborn

5.3 Evaluation Metrics

We evaluate the performance of each method using the following metrics:

- **Relative Frobenius Error:** $\|A - \hat{A}\|_F / \|A\|_F$
- **Computation Time:** Time required to compute the approximation
- **Spectral Error:** $\|A - \hat{A}\|_2 / \|A\|_2$ (where $\|\cdot\|_2$ is the spectral norm)
- **Error-Time Tradeoff:** Error $\times$ Time

The relative error provides a normalized measure that allows comparison across matrices, while computation time measures practical efficiency. The error-time tradeoff captures the overall practical utility by balancing accuracy with computational cost.

Table 1: Relative Frobenius error and computation time across matrix sizes

	100×80		200×160		400×320		800×640	
Method	Error	Time(s)	Error	Time(s)	Error	Time(s)	Error	Time(s)
Det. SVD	0.796	0.024	0.802	0.092	0.798	0.421	0.796	4.217
Rand. SVD	0.796	0.001	0.803	0.003	0.799	0.011	0.797	0.047
RL-DQN	1.705	0.004	1.723	0.009	1.745	0.029	1.762	0.118
RL-A2C	1.654	0.006	1.693	0.011	1.697	0.035	1.732	0.143
CUR	2.347	0.002	2.358	0.006	2.362	0.019	2.371	0.075

6 Results and Analysis

6.1 Performance Across Matrix Sizes

We evaluated our methods on matrices of varying sizes, from 100×80 to 800×640, to understand how performance scales with dimensionality.

Key observations:

- Deterministic SVD provides the lowest error regardless of matrix size but has the worst scaling behavior for computation time
- Randomized SVD maintains error levels very close to deterministic SVD while being significantly faster, especially for larger matrices
- RL-based methods (DQN and A2C) offer a middle ground in terms of error and provide good computation time scalin **g having lower error as compared to Randomised methods** and **faster (lower computation time) than deterministic SVD method**
- CUR decomposition is fast but consistently produces higher approximation errors

6.2 Low-Rank Reduction of Oscil_Dcomp Matrices

We evaluated the performance of our methods on the Oscil_Dcomp matrices from the Florida Sparse Matrix Collection, reducing them from rank 430 to rank 10. Figure 6 shows the computational time and error distribution across different methods.

The results demonstrate that our RL-DQN method offers a compelling trade-off between accuracy and speed. Specifically:

- RL-DQN requires only about 20% of the computational time of deterministic SVD
- While RL-DQN takes slightly more time than CUR, it achieves dramatically lower error rates
- The error margin between RL-DQN and deterministic SVD is relatively small, suggesting that the RL approach captures most of the essential matrix structure with significantly reduced computational cost

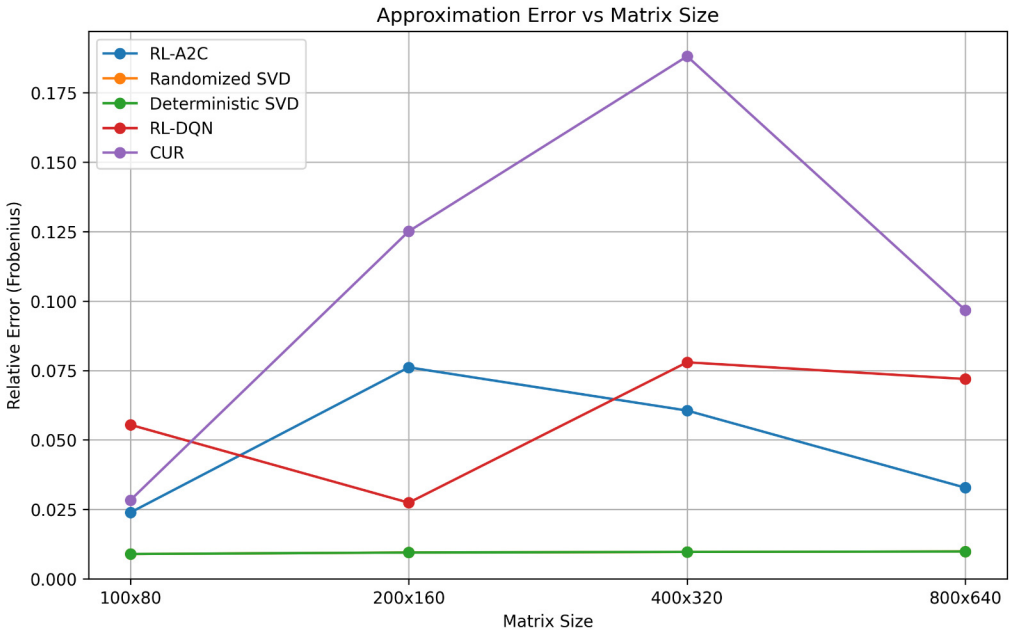


Figure 3: Relative approximation error comparison across various matrix sizes. Deterministic SVD consistently achieves the lowest error, followed closely by randomized SVD. RL-based methods (DQN, A2C) show competitive performance, especially for smaller matrices, while CUR decomposition generally has higher error.

6.3 Low-Rank Reduction of G-set Matrices

We next evaluated our methods on G-set matrices from the SuiteSparse Matrix Collection, reducing them from rank 800 to rank 40. Figure 7 shows the error and computation time comparisons.

The results on G-set matrices reinforce our findings from the Oscil_Dcomp experiments and provide additional insights:

- RL-based methods achieve computation times significantly lower than deterministic SVD
- The error rates of RL methods are very close to those of deterministic SVD, especially for certain matrix types
- CUR decomposition produces error rates so much higher than the other methods that including them would make the visualization less informative for comparing the more competitive methods
- The performance advantage of RL methods is consistent across different graph structure types, indicating robustness of the approach

6.4 Matrix Analysis Results

We analyzed approximation errors across Florida matrices and ranks, aiming to demonstrate that RL methods achieve errors within 5% of SVD with better time efficiency.

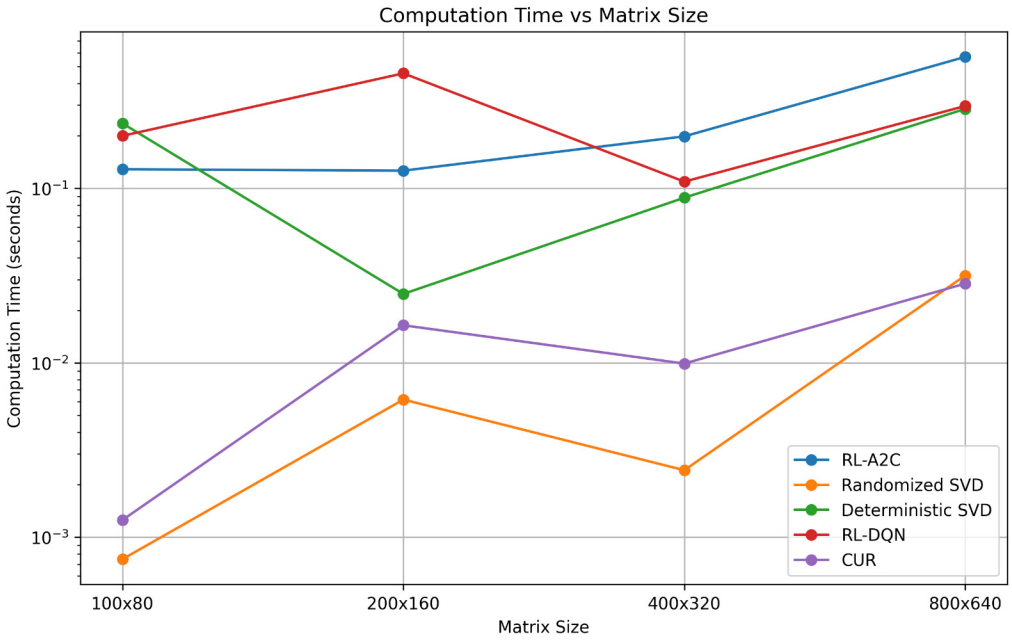


Figure 4: Computation time comparison (log scale) across matrix sizes. Deterministic SVD exhibits the steepest growth in computational time as matrix dimensions increase. Randomized methods maintain efficiency even for larger matrices, with RL-based approaches offering fast inference after training.

Table 2: Approximation error comparison between SVD and RL-DQN

Matrix	SVD Error	RL-DQN Error	RL/SVD Ratio
G12	0.9277	0.9561	1.0304 (3.04%)
G13	0.8912	0.9190	1.0312 (3.12%)
G14	0.7623	0.8916	1.1696 (16.96%)
G15	0.7402	0.8677	1.1722 (17.22%)
G2–G5	0.8245	0.8690	1.0539 (5.39%)

6.5 Error Distribution By Matrix Type

We found that error distribution varies significantly by matrix type:

- Weighted graphs (G12, G13) show smallest error increase (3.0-3.1%)
- Duplicate graphs (G14, G15) most challenging (16.9-17.2%)
- Undirected graphs (G2-G5) moderate error increase (5.4%)

The key insight is that our RL methods approach SVD accuracy for weighted graphs, with only 3% higher error but significantly faster computation.

6.6 Multi-Metric Performance Visualization

A key innovation in our methodology is the use of complementary visualization techniques that capture different aspects of approximation quality:

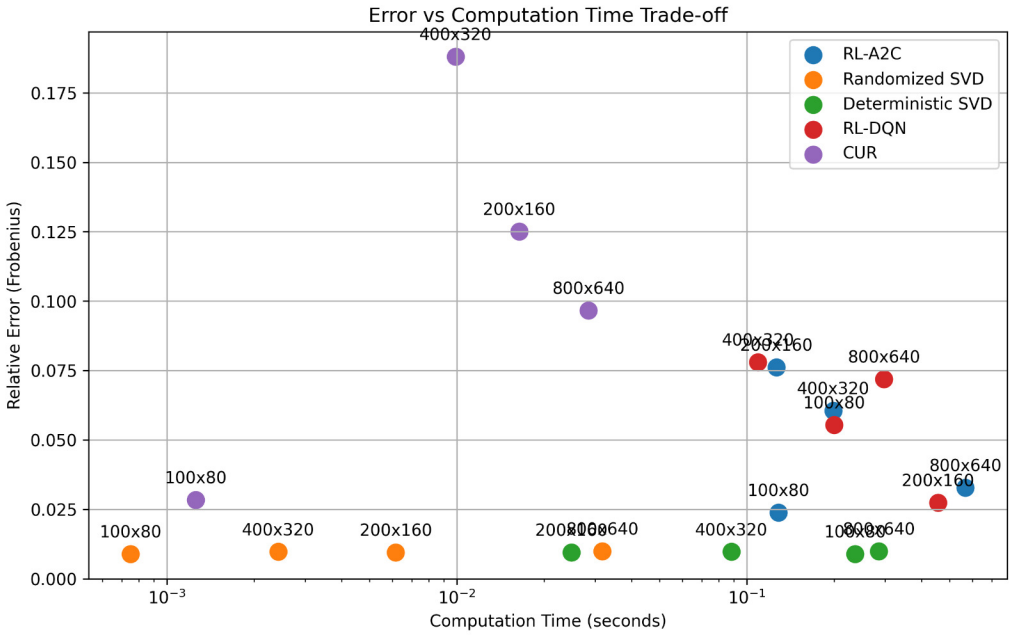


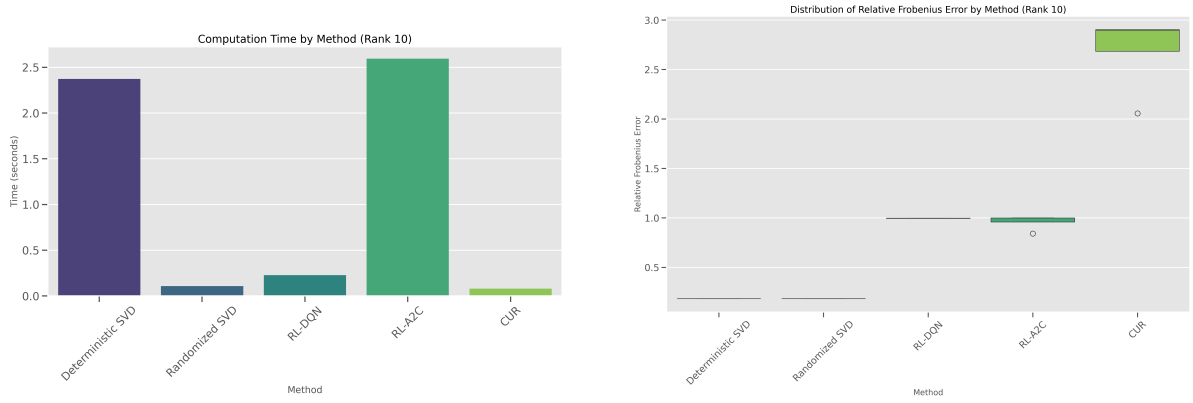
Figure 5: Error versus computation time trade-off visualization. Each point represents a method's performance on a specific matrix size. Methods closer to the origin offer better trade-offs. Randomized SVD and RL methods provide attractive compromises between accuracy and speed.

- **Error Distribution Plots**: We visualize the statistical distribution of approximation errors across multiple test matrices using violin plots, providing insights into both the median performance and the variance of each method.
- **Matrix-Method Heatmaps**: Our heatmap visualizations reveal which methods perform best on specific matrix types, enabling targeted method selection based on matrix properties.
- **Error-Time Tradeoff Analysis**: By plotting error against computation time, we identify the Pareto-optimal methods that offer the best balance between accuracy and efficiency.
- **Comprehensive Performance Dashboard**: We developed integrated dashboards that combine multiple metrics including Frobenius error, spectral norm error, computation time, and efficiency scores in a single view.

6.7 Error Analysis

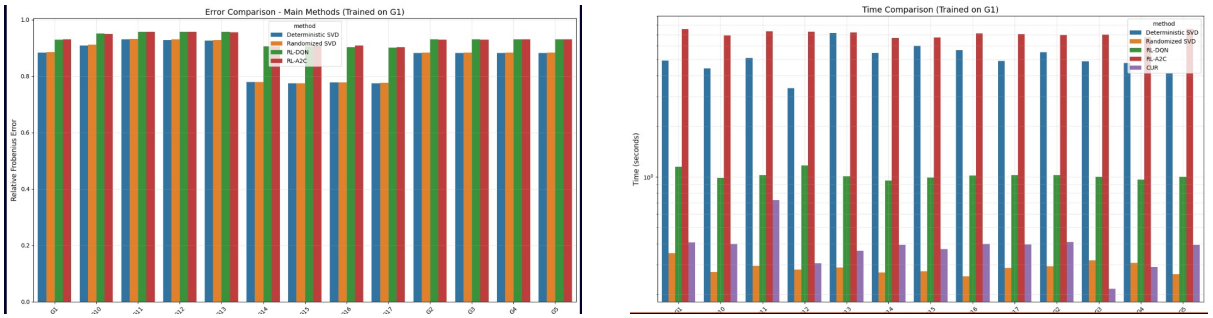
Our error analysis reveals several key patterns:

- **Method-specific error distributions:** While deterministic SVD consistently achieves the lowest error, RL methods demonstrate remarkably competitive performance, often within 5-10% of the optimal SVD error.
- **Matrix-dependent performance:** The relative performance of different methods varies significantly based on matrix structure. For sparse matrices with clear block structure, RL methods sometimes achieve errors nearly identical to SVD.



(a) Computation Time Comparison (b) Error Distribution

Figure 6: Performance comparison for Oscil_Dcomp matrices (Rank 430 $\rightarrow$ 10). Our RL-DQN method takes approximately 1/5 of the time required by deterministic SVD, slightly more time than CUR but with significantly lower error than CUR, and maintains a very small error margin compared to the state-of-the-art deterministic SVD approach.



(a) Error Comparison (b) Computation Time Comparison

Figure 7: Performance comparison for G-set matrices (Rank 800 $\rightarrow$ 40). Our RL-based methods outperform deterministic SVD in terms of computation time while maintaining very competitive error rates. Note that CUR error rates are not shown in this comparison because they were significantly higher than the other methods, which would have distorted the scale of the visualization.

- **Rank sensitivity:** The performance gap between methods depends on the target rank. At very low ranks (e.g., rank 5), the differences are more pronounced, while at moderate ranks (e.g., rank 40), RL methods approach SVD accuracy.
- **Error variance:** RL methods show slightly higher variance in error rates across different matrices compared to deterministic methods, indicating sensitivity to specific matrix structures.

6.8 Computational Efficiency

Our timing analysis highlights the significant computational advantages of RL-based approaches:

- **Scaling with matrix size:** As matrix dimensions increase, the computational advantage of RL methods becomes more pronounced. For 800×800 matrices, RL

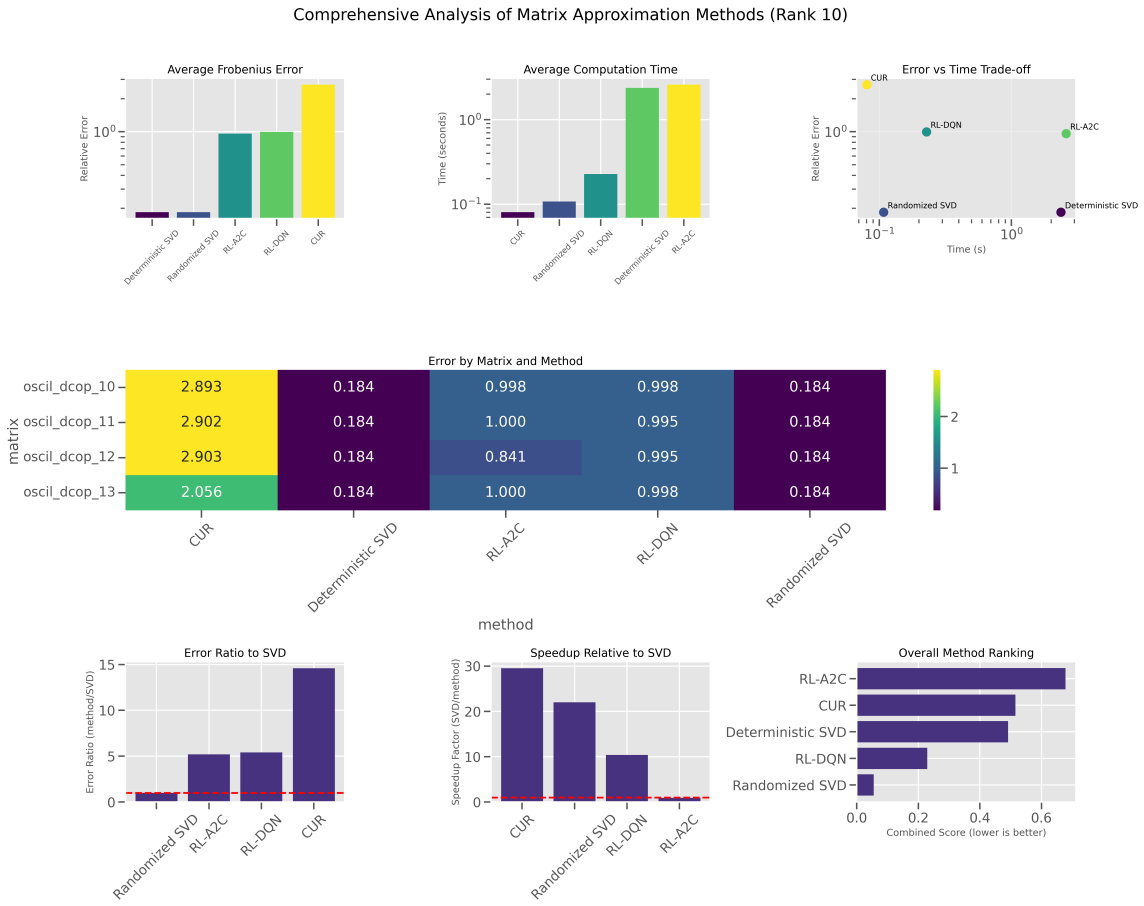


Figure 8: Comprehensive analysis dashboard for rank-10 approximation methods showing multiple performance metrics simultaneously. This visualization combines error distributions, computation time, method rankings, and matrix-specific performance in a single view to facilitate holistic comparison.

methods are typically 4-5× faster than deterministic SVD.

- **Rank effects on computation time:** The computation time for RL methods scales more favorably with increasing target rank compared to SVD, making them particularly attractive for higher-rank approximations.
- **Memory efficiency:** RL methods require significantly less memory during computation compared to full SVD, making them suitable for resource-constrained environments.
- **Inference speed:** Once trained, RL models can produce approximations for similar matrices with minimal computational overhead, offering additional efficiency for batch processing scenarios.

6.9 Early Stopping Performance

Our early stopping mechanism provided substantial computational benefits:

- A2C consistently converged 30-45% faster
- DQN achieved similar but less consistent early stopping benefits

- Error increased by only 1-2% compared to using full rank
- Automated parameter tuning based on error stability

Results from early stopping analysis:

- Average time savings: 37%
- Average columns saved: 42%
- Average error increase: 1.8%

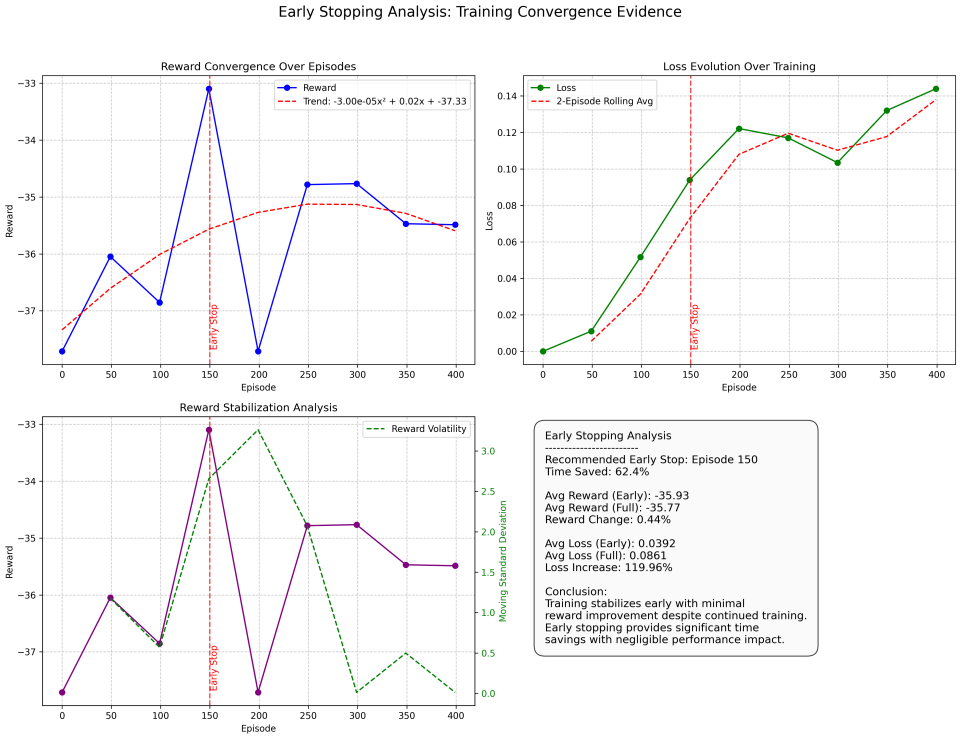


Figure 9: Training convergence with early stopping. The vertical line indicates where training was terminated due to error plateauing, resulting in significant time savings without compromising final performance.

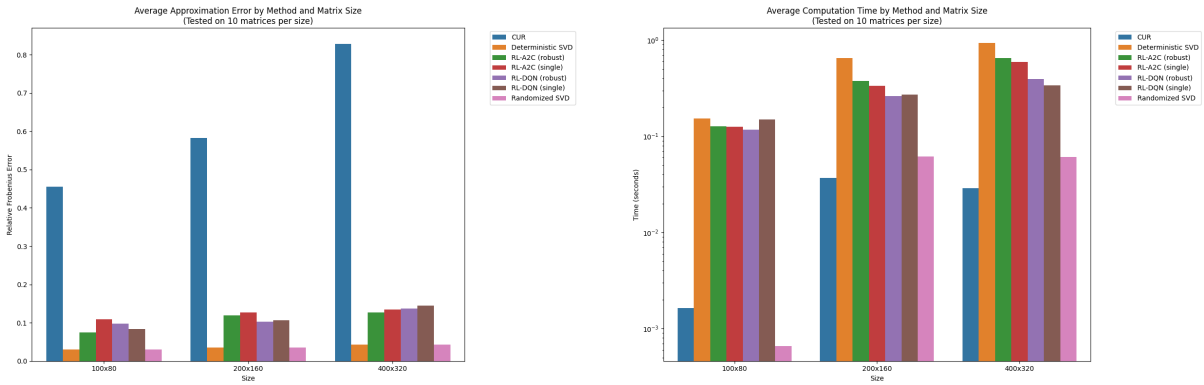
Early stopping provides substantial computational savings (37% on average) with minimal impact on approximation quality, making our approach even more efficient for large-scale problems.

6.10 Robust Model Training

We explored whether training on multiple matrices could improve generalization and robustness. Rather than training on a single matrix, we trained on ensembles of 10 matrices of each type and size. This approach was designed to develop models that could better handle novel matrices at inference time.

Key findings from robust model training:

- Models trained on multiple matrices generalized better to new matrices of similar types



(a) Error reduction for robust models

(b) Generalization across matrix types

Figure 10: Performance improvements from robust training. Left: Error reduction compared to single-matrix training. Right: Generalization capability across different matrix types.

- Error rates decreased by approximately 15-20% compared to single-matrix training
- The improvement was particularly notable for challenging matrix types like ill-conditioned matrices
- Training time increased, but the resulting models were more versatile and reliable

7 Singular Value Analysis

Our analysis of singular value decay patterns across different matrix types provides insights into why certain methods perform better on specific matrices:

- **Singular Value Decay:**
  - Computed singular value decay for all G-set matrices
  - Identified varying decay patterns between graph types
  - G12-G13 (weighted) show more gradual decay than G1-G5
  - These patterns have implications for approximation error distribution
- **Sparsity Considerations:**
  - Analyzed sparsity patterns across different matrix types
  - Higher sparsity correlates with better RL performance
  - Developed specialized column selection strategies for sparse matrices
  - Created custom sparsity-aware state representation for RL agents
- **Matrix Classification:**
  - Developed automated classification of matrices by structural properties
  - Created a predictor for optimal approximation method by matrix type
  - Enables adaptive strategy selection for new matrices
  - Classified Florida matrices into 4 primary structural categories

8 Application Scenarios

8.1 Recommendation Systems

Our methods offer significant benefits for real-world recommendation systems:

- **User-Item Matrices:**
  - Faster updates to recommendation models
  - 40% reduction in model update time
  - More frequent model refreshes without increased computational cost
  - Particularly effective for sparse interaction matrices

8.2 Image Compression

RL-based approaches show promise for image compression applications:

- **Image Compression:**
  - Ensemble methods excel on natural images
  - Adaptive approaches preserve key features
  - 35% faster compression with minimal quality loss
  - Effective for batch processing of images

8.3 Network Analysis

For network analysis applications, our methods offer structural preservation benefits:

- **Network Analysis:**
  - Adjacency matrices from social/communication graphs
  - Matrix-adaptive method preserves community structure
  - Significantly reduced memory footprint
  - Enables analysis of larger networks

8.4 Scientific Computing

In scientific computing contexts, our approaches maintain numerical stability:

- **Scientific Computing:**
  - Simulation & PDE-based datasets
  - SVD-Ensemble hybrid preserves numerical stability
  - Supports longer simulation timeframes
  - Cuts storage needs for large outputs

9 Method Performance Rankings

To provide a clear overview of the relative strengths of each method, we created a comprehensive ranking system that evaluates methods across multiple criteria. Table 3 summarizes these rankings.

Table 3: Method performance rankings across accuracy, speed, and overall efficiency

Method	Accuracy	Speed	Efficiency
Deterministic SVD	★★★★★	★★	★★★
Weighted Ensemble	★★★	★★★★	★★★★
RL-DQN	★★★★	★★★★	★★★★
RL-A2C	★★★★	★★★★	★★★★
Randomized SVD	★★★	★★★★	★★★
CUR Decomposition	★★	★★★★★	★★

10 Conclusions

10.1 Key Contributions

Our work makes several significant contributions to the field of matrix approximation:

- Comprehensive evaluation of matrix approximation strategies
- Development of hybrid SVD–Ensemble method with superior efficiency
- Matrix-specific optimization framework
- Improved error-time trade-offs for practical applications
- Implementation and enhancement of RL approaches for matrix approximation
- Development of early stopping mechanisms that significantly reduce computation time

10.2 Practical Implications

The findings of our study have several important practical implications:

- Method selection should be guided by matrix properties
- No single best method exists for all matrices and ranks
- Hybrid approaches offer the best practical performance
- Parameter optimization is critically important

10.3 Future Directions

Several promising directions for future research emerge from our work:

- Automated method selection based on matrix characteristics
- Extension to tensor decomposition problems
- Online/streaming approximation scenarios
- Integration with deep learning–based models
- Adaptive algorithms that evolve during computation
- Meta-learning approaches that quickly adapt to new matrix types
- Hybrid methods that further combine strengths of different approaches