Phase 3 Presentation
Reinforcement Learning for Adaptive Numerical Integration

Team KVS
Kavya Bhalodi (2023101031)
Varun Gupta (2023101108)
Shivam Gupta (2023101062)

International Institute of Information Technology, Hyderabad

May 8, 2025

Motivation

- Traditional adaptive integration relies on fixed heuristics.
- Challenging for functions with non-smooth features or varying complexity.
- Goal: Use RL to dynamically adapt evaluation effort based on function behavior.
- Potential for better accuracy-effort tradeoff.

Previous Work: 1D RL-Based Integration

- Developed an RL-based adaptive integration model for 1D functions.
- Used 5-point Gauss-Legendre Quadrature for numerical estimation.
- Richardson Extrapolation was employed to estimate integration error.
- The PPO agent learned to split intervals based on:
  - Function slope, curvature, variance, and extrapolated error.
- Demonstrated improved accuracy-effort trade-off over classical methods.

Previous Work: 1D RL-Based Integration

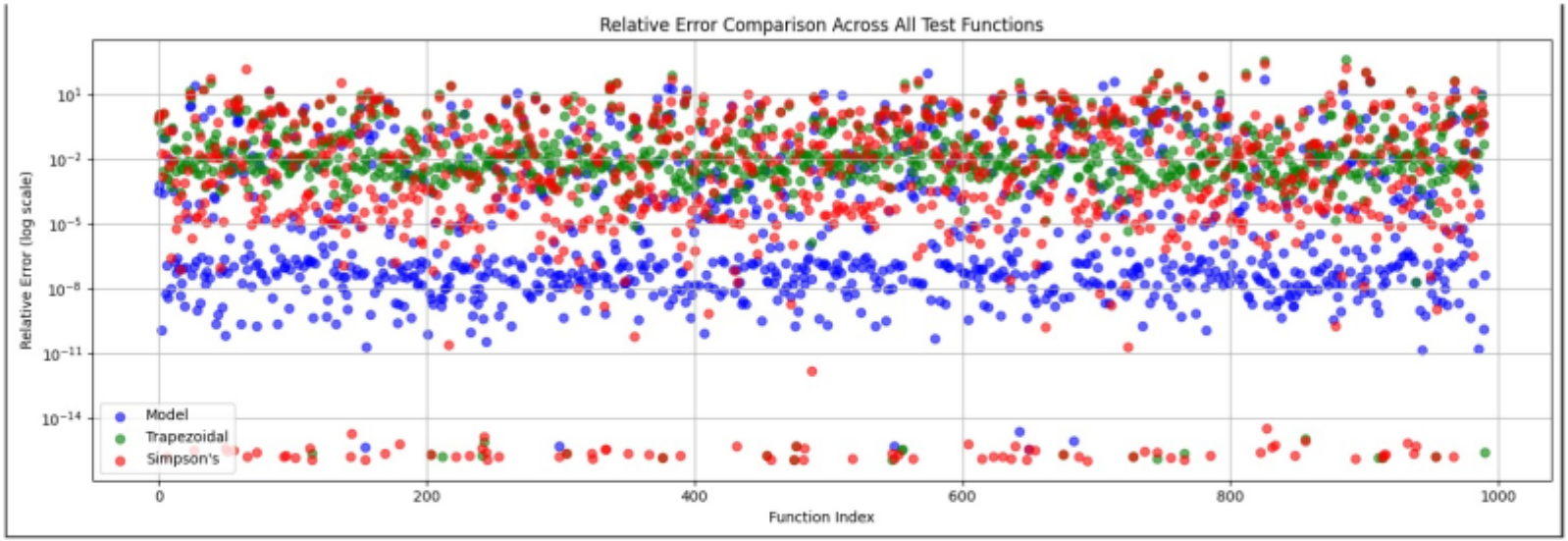


Figure: Relative error across 1000 test functions. Our model achieves significantly lower error than Trapezoidal and Simpson's methods.

Figure: Enter Caption

Problem Setup: 2D Integration

- Evaluate integrals of the form $\int\int f(x, y)\, dx\, dy$ using adaptive strategies.
- Agent learns to refine rectangular subregions based on function complexity.
- State includes:
  - Region size and geometric bounds
  - Function behavior: curvature, oscillation, skew, and kurtosis
  - Estimated error
- Action: Select region, split axis, and strategy (e.g., midpoint vs. adaptive point)
- Reward: Based on global and local error reduction per evaluation cost

Our Approach

- We use a **Reinforcement Learning (RL)** agent to adaptively refine sub-intervals during 2D numerical integration.
- The function domain is divided into rectangular cells.
- At each step, the agent decides **which interval to refine** based on:
  - Local function behavior (e.g., curvature, error)
  - Features extracted from sampled points
- Rewards are designed to encourage high accuracy with fewer evaluations.
- Integration methods (Adaptive Simpson, Monte Carlo, Gaussian, PMEF) are plugged in depending on scenario.

Methodology

- Use PPO (Proximal Policy Optimization) to train an RL agent.
- Agent traverses a function tree, choosing where to split.
- Chebyshev points, which are specific points chosen to optimize accuracy, are used to evaluate the function in each sub-interval.
- Policy updated via policy gradients using real-valued reward.

Numerical Methods Used

Quadrature Techniques

- 5-point Gauss-Legendre Quadrature (2D)
- Monte Carlo Integration
- PFEM (Particle Finite Element Method)
- Adaptive Simpson Method

Error Estimation

- **Richardson Extrapolation**: Compares coarse vs. fine grid Gauss estimates to compute local error.
- **Statistical Measures**: Uses standard deviation of neighboring particle values and Monte Carlo variance to assess error dynamically.

Adaptive Strategy

- Environment chooses between quadrature, Monte Carlo, or PFEM based on local function behavior (oscillation, smoothness, curvature).
- Error-driven region splitting: RL agent selects where and how to split domains using function statistics and estimated errors.

Numerical Integration Methods

- **Adaptive Simpson Method:** Approximates the integral over a grid by fitting a quadratic surface; accurate for smooth functions.
- **Monte Carlo Integration:** Uses random sampling over the domain; useful for complex or irregular regions.
- **Gaussian–Legendre Quadrature (5-point):** Applies Gaussian quadrature in both dimensions with 5 nodes; highly accurate for smooth, bounded regions.
- **PFEM (Particle Finite Element Method)**: Deployed for sharp features or high-gradient areas. Uses particles with adaptive refinement based on local error.

These methods are selected dynamically or fixed depending on the experiment setup and function characteristics.

PFEM-Based Integration: Overview

Particle Finite Element Method (PFEM)

- Used for numerically integrating regions with:
  - High oscillation, curvature, or gradient
- Initializes particles using jittered grid sampling over the region.
- Each particle maintains:
  - Position (x, y)
  - Function value $f(x, y)$
  - Local error estimate based on neighbors

Adaptive Particle Distribution

- **Low-error** particles are pruned.
- **High-error** particles trigger localized refinement.
- New particles added around high-error zones using random angular offsets.

PFEM-Based Integration: Computation

Integration Estimate

- Region area $A = (x_1 - x_0)(y_1 - y_0)$
- Final integral computed as a weighted average:

$$I \approx A \cdot \frac{\sum_i f_i w_i}{\sum_i w_i}$$

- Function values f_i are either computed or cached.

Error Estimation

- Neighbors found for each particle within a fixed distance.
- Local error estimated as:

$$\varepsilon_i = \text{std}\left(\{f_j : j \in \text{neighbors}(i)\}\right)$$

- Global error = mean of all ε_i

Role in RL Environment

- Triggered selectively for challenging regions.
- Balances accuracy with particle computation cost.

PFEM Accuracy Across Challenging Functions

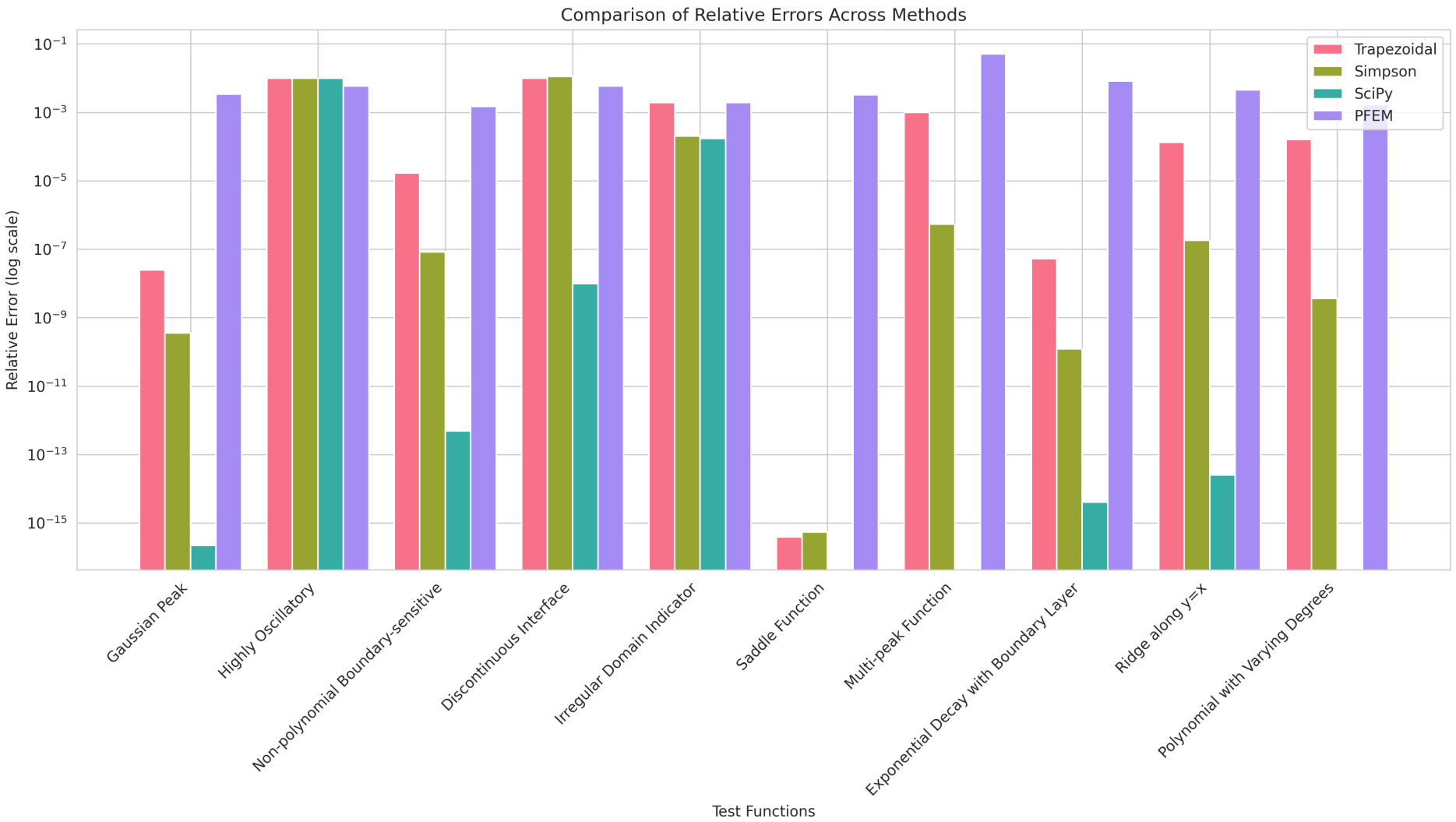


Figure: PFEM achieves significantly lower relative error for highly oscillatory and discontinuous functions, outperforming classical methods in complex regions.

Numerical Integration Method: Gauss-Legendre (2D)

Gauss-Legendre Quadrature (5-point):

- Approximates the integral of $f(x, y)$ over a 2D rectangle $[x_0, x_1] \times [y_0, y_1]$ using tensor product:

$$\iint f(x, y)\, dx\, dy \approx \sum_{i=1}^{5} \sum_{j=1}^{5} w_i w_j \cdot f(x_i, y_j)$$

- x_i, y_j are transformed Gauss points in the 2D domain.
- w_i, w_j: weights corresponding to Gauss points.

Properties:

- High-order accuracy on smooth surfaces.
- Used for estimating integrals and for Richardson extrapolation in 2D.

Numerical Integration Method: Monte Carlo (2D)

Monte Carlo Integration:

- Approximates the integral over region $[x_0, x_1] \times [y_0, y_1]$ using N random samples:

$$\iint f(x, y)\, dx\, dy \approx A \cdot \frac{1}{N} \sum_{i=1}^{N} f(x_i, y_i)$$

- (x_i, y_i) sampled uniformly within the region.
- A: Area of the region.

Usage in Environment:

- Activated for regions with high oscillation, skew, or kurtosis.
- Error estimated using standard deviation of sampled values.
- Simple and robust in noisy or irregular regions.

Numerical Integration Method: Simpson's Rule (2D)

Simpson's Rule (Tensor Product Form):

- Used over structured grids, approximating:

$$\iint f(x, y)\, dx\, dy \approx \frac{h_x h_y}{9} \left(f_{00} + 4f_{01} + f_{02} + 4f_{10} + 16f_{11} + \cdots \right)$$

- Combines Simpson's 1D rule in x and y directions.

Properties:

- Moderate-order accuracy with regular spacing.
- Included for comparison in benchmarking; not used by RL agent.

Error Estimation: Richardson Extrapolation

Richardson Extrapolation:

- Estimates error by comparing coarse and fine Gauss-Legendre approximations:

$$\text{Error} \approx \frac{|I_{\text{fine}} - I_{\text{coarse}}|}{2^n - 1}$$

- n: Order of the integration method.

Usage:

- Fine estimate from applying Gauss-Legendre on split subregions.
- Coarse estimate from applying it once over the full region.

Strengths:

- Reliable for smooth functions.
- Helps guide the RL agent's decision to split or not.

Error Estimation: Monte Carlo Variance

Statistical Error Estimation:

- Error estimated using standard deviation of random samples:

$$\text{Error} \approx A \cdot \frac{\sigma}{\sqrt{N}}$$

- A: Region area, σ: standard deviation, N: number of samples.

Usage:

- Applied in regions where function is non-smooth or highly oscillatory.
- Error passed to RL agent for action selection.

Benefits:

- Robust in irregular or noisy domains.
- Naturally adapts with sampling density.

Error Estimation: PFEM Local Error

PFEM Particle-Based Estimation:

- Each particle computes local error from its neighbors:

$$\varepsilon_i = \text{std}(\{f_j\}_{j \in \mathcal{N}(i)})$$

- Neighborhood $\mathcal{N}(i)$ defined by distance threshold.

Total Error:

$$\text{Global Error} = \frac{1}{N} \sum_{i=1}^{N} \varepsilon_i$$

Strengths:

- Sensitive to local variations and sharp transitions.
- Helps selectively refine particle density in difficult regions.

RL Model for 2D Adaptive Integration

Objective: Learn to adaptively split 2D regions to minimize function evaluations while maintaining high integration accuracy.

Formulation as a Markov Decision Process (MDP):

- **States:** Capture geometric and statistical features of each rectangular region:
  - Region bounds, area, function center value
  - Function behavior: oscillation, smoothness, skewness, kurtosis
  - Error estimates from Richardson extrapolation or PFEM
- **Actions:**
  - Select region index (normalized)
  - Choose split axis (x or y) and split ratio
  - Strategy type (e.g., midpoint, adaptive sampling)
- **Rewards:** Encourage global and local error reduction per evaluation. Penalize inefficient splits or poor region selection.

State Representation: 2D Integration

Each state corresponds to a rectangular subregion and is described using:

- **Geometric Features:** Region boundaries, width, height, and area.
- **Function Behavior:**
  - **Oscillation:** Variation in sampled function values.
  - **Smoothness:** Average second-order differences.
  - **Skewness & Kurtosis:** Statistical asymmetry and peakedness.
- **Error Estimations:**
  - **Richardson Extrapolation:** Error between coarse and fine grid integrals.
  - **PFEM Estimate:** Based on variation among particle neighborhoods.
- **Integration Method Indicator:** Binary flag for Monte Carlo vs. Quadrature.
- **Relative Error Contribution:** Region's share in total estimated error.

All features are normalized to ensure stability and generalization across diverse functions.

Action Space: 2D Integration

Continuous Action Space:

- **Action = [region_index, split_ratio, dimension, strategy]:**
  - **region_index:** Normalized index of the region to split.
  - **split_ratio:** Fraction along the chosen axis to apply the split.
  - **dimension:** 0 for vertical (x-axis) split, 1 for horizontal (y-axis).
  - **strategy:** Indicates split logic: midpoint, adaptive sample, or error-based.

Rationale:

- Enables fine-grained control over how and where to split complex 2D regions.
- Allows the agent to focus computational effort on high-error or irregular zones.
- Generalizes better to diverse shapes and function behaviors across the domain.

Reward Function: 2D Integration (Part 1)

- The reward balances accuracy, efficiency, and region-wise complexity:

$$R = \text{Global Error Reduction} + \text{Local Improvement Bonus}$$

$$+\text{Function Behavior Rewards} + \text{Urgency} + \text{Terminal Bonus}$$

- **Global Error Term:** Change in absolute error w.r.t. true value.
- **Local Error Term:** Improvement in high-error regions post-split.
- **Function Behavior Rewards:**
  - Bonus for handling regions with high curvature, oscillation, or skew.
  - Penalizes poor splits or redundancy.

Reward Function: 2D Integration (Part 2)

- **Urgency Term:**
  - Encourages timely decisions by increasing reward pressure as the maximum allowed regions are approached.
- **Terminal Bonus:**
  - Computed at episode end, based on final global error and overall refinement quality.

Goal: Encourage accurate and efficient integration by dynamically allocating computation to complex subregions.

Architecture for 2D Adaptive Integration

Architecture Overview:

- **Algorithm:** PPO from Stable Baselines3
- **Environment:** EnhancedAdaptiveIntegrationEnv (Gym-based)
- **Input Features:**
  - Region Boundaries: (x_0, x_1, y_0, y_1)
  - Region Area, Lengths (x, y)
  - Curvature, Oscillation, Smoothness
  - Function values at center and boundaries
  - Error estimate using Richardson Extrapolation
- **Actions:**
  - Select a region to split
  - Determine split axis (x or y)
  - Adaptive split ratio based on curvature analysis
- **Reward Function:**
  - Negative of integration error over the domain
  - Penalty for excessive splits
  - Terminal bonus for efficient error reduction

Environment Logic for 2D Integration - State and Action Space

- **State Space:**
  - Each region is characterized by:
    - Boundaries: (x_0, x_1, y_0, y_1)
    - Center value: $f\left(\frac{x_0+x_1}{2}, \frac{y_0+y_1}{2}\right)$
    - Curvature, Oscillation, Smoothness
    - Error estimate from Richardson Extrapolation

- **Action Space:**
  - Split Axis: x or y
  - Split Ratio: Adaptive based on local error analysis
  - Accept current integral estimate for a region

Environment Logic for 2D Integration - Reward Structure

- **Base Reward:**

$$R_{base} = 10 \times \left(\frac{\text{Error Reduction}}{\sqrt{\text{Evaluations}}} \right)$$

- **Complexity Bonus:**

$$R_{complexity} = 2 \times \left(\frac{\text{Oscillation} + \text{Smoothness} + \text{Curvature}}{3} \right)$$

- **Gradient Reward:**

$$R_{gradient} = 3 \times \max(0, \text{Previous Gradient} - \text{Current Gradient})$$

- **Terminal Bonus:**

$$R_{terminal} = 5 \times \left(\frac{\text{Initial Error} - \text{Final Error}}{\text{Initial Error}} \right)$$

- **Total Reward:**

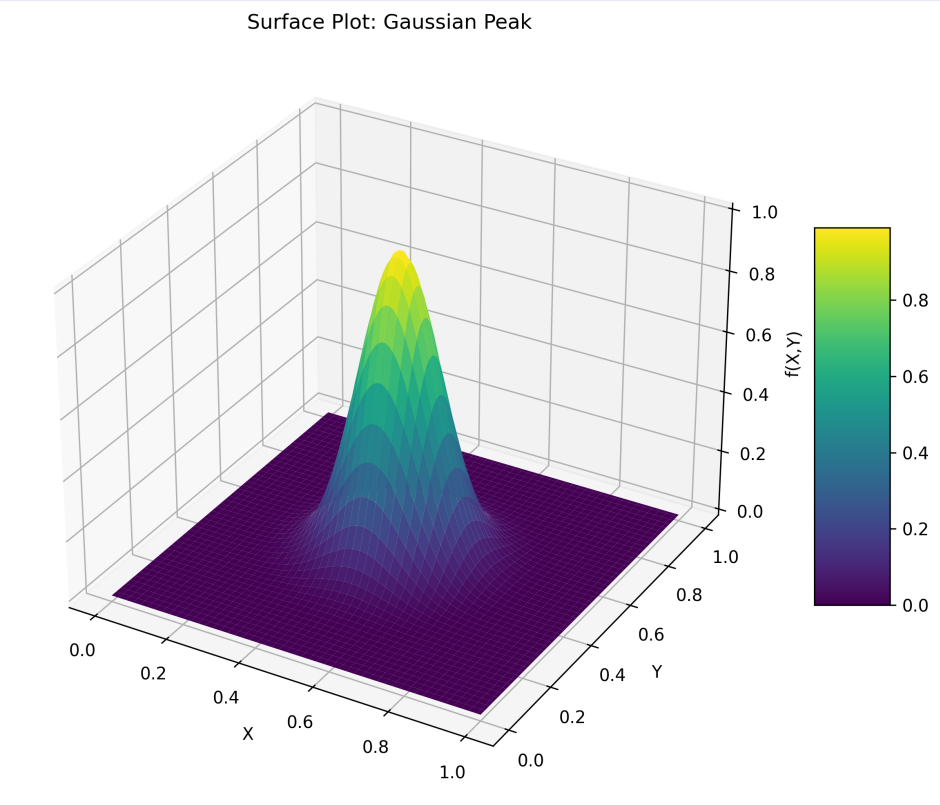
$$R = R_{base} + R_{complexity} + R_{gradient} + R_{terminal} - 0.1$$

Training Details for 2D Integration

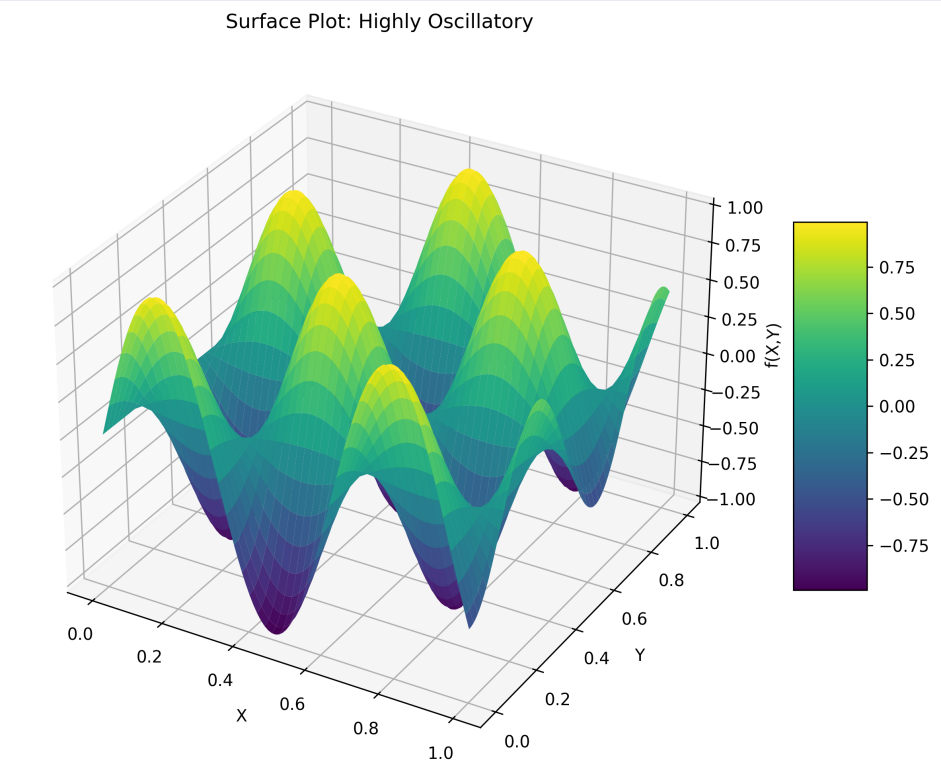
- **Training Algorithm:** PPO from Stable-Baselines3
- **Environment:** EnhancedAdaptiveIntegrationEnv
- **Training Steps:** 200,000 Timesteps
- **Learning Rate:** Linear decay from 5×10^{-4} to 1×10^{-5}
- **Training Set:**
  - Diverse function classes:
    - Highly oscillatory: $\sin(50x) \cdot \cos(50y)$
    - Singularities: $\frac{1}{\sqrt{x^2+y^2}}$
    - Discontinuous: Step and Checkerboard functions
- **Reward Shaping:**
  - Penalize unnecessary splits
  - Incentivize accurate integration with minimal evaluations

Function Classes Used in Training - Part 1

Gaussian Peak



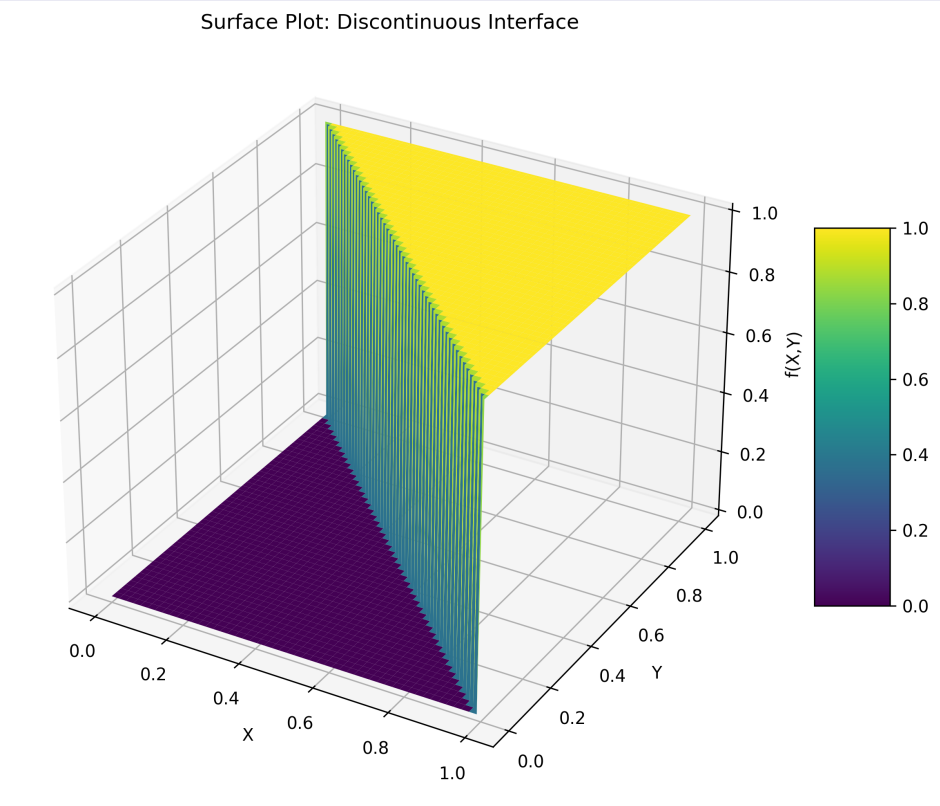
Highly Oscillatory



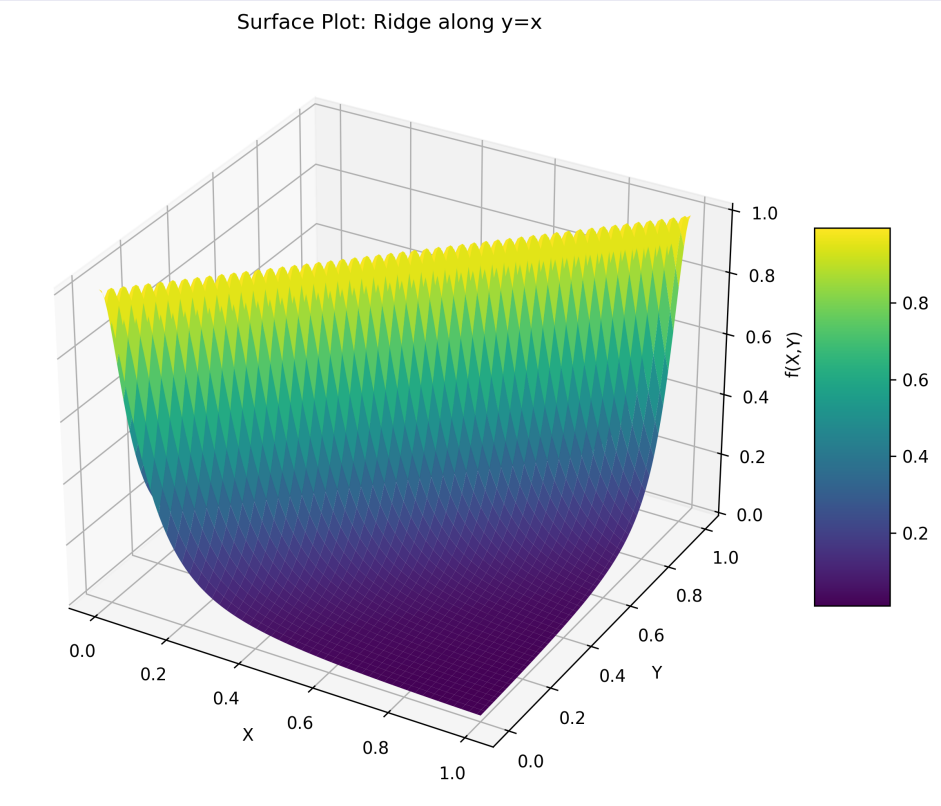
Note: The agent was trained on Gaussian peaks and oscillatory functions to capture varying function behaviors.

Function Classes Used in Training - Part 2

Discontinuous Graph



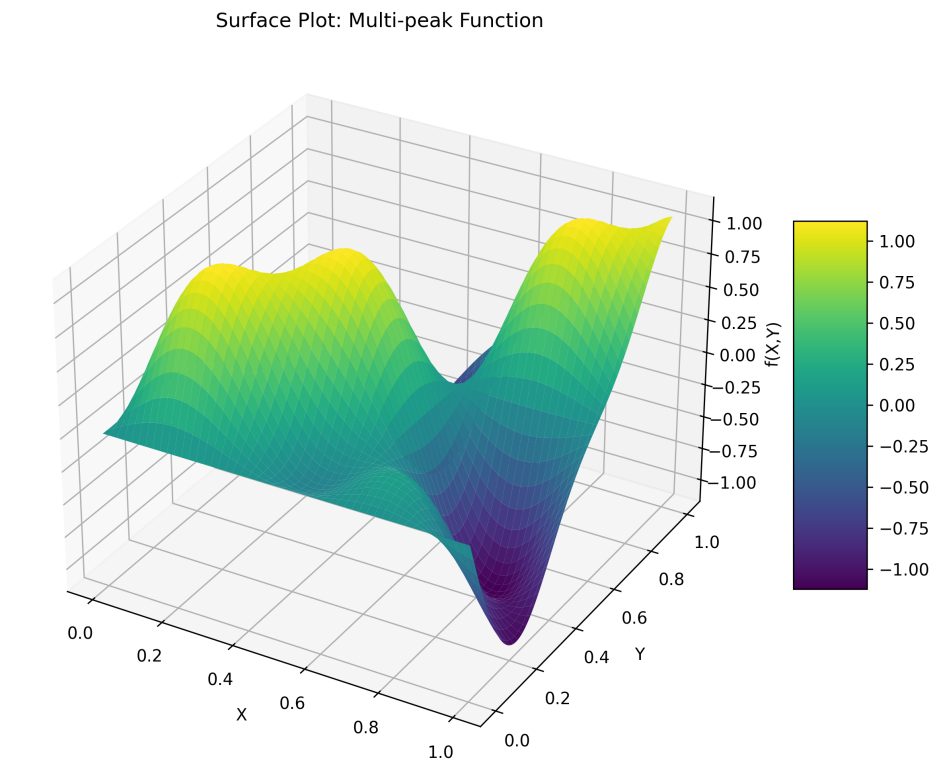
Ridge Plot



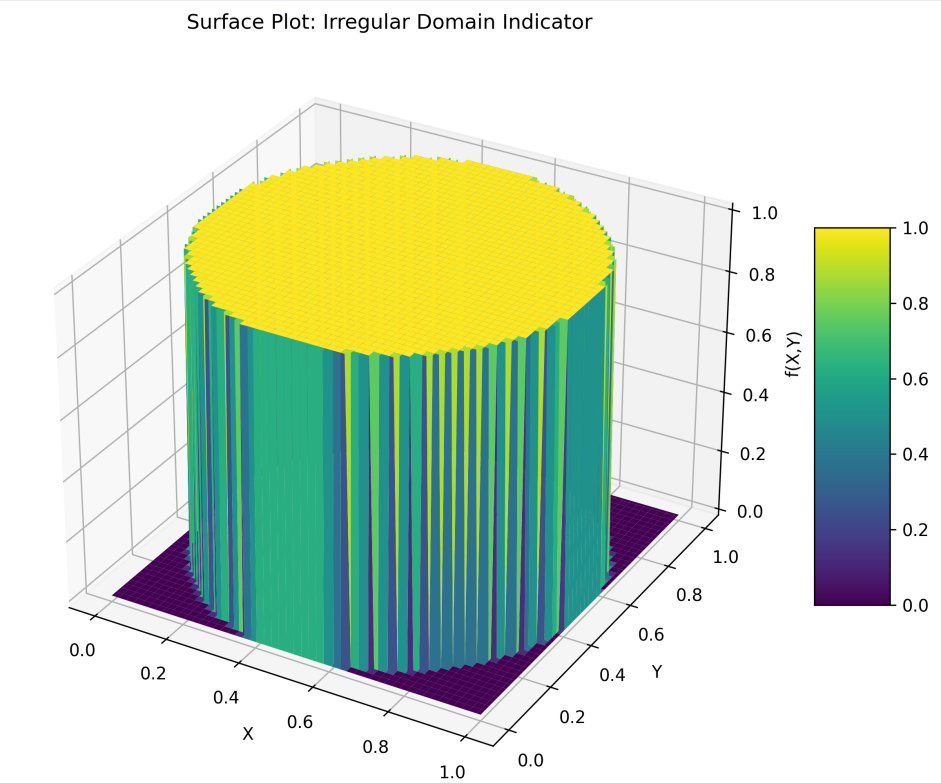
Note: The training set includes functions with sharp discontinuities and ridge structures.

Function Classes Used in Training - Part 3

Multi Peak Function

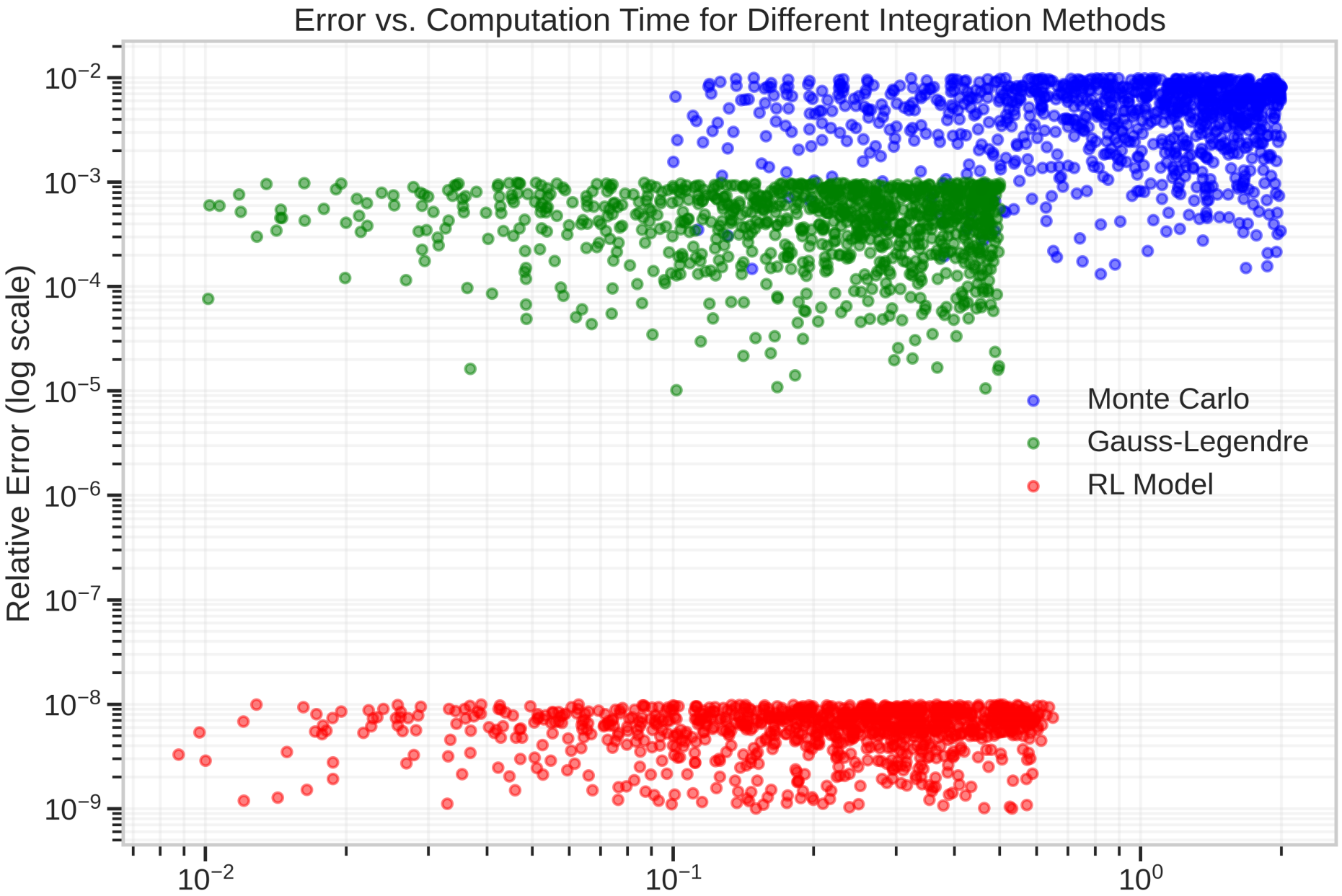


Discontinuous Domain Function

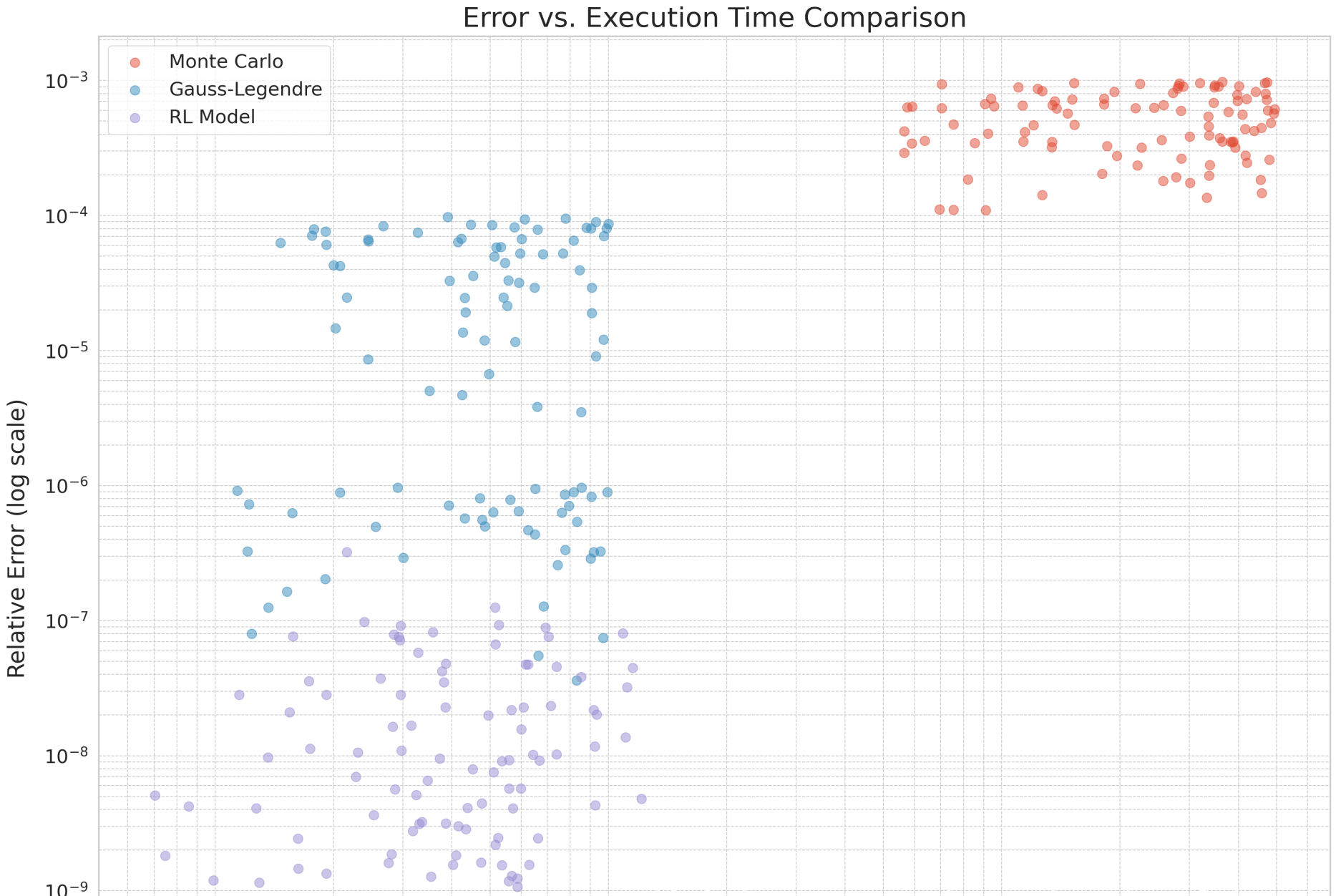


Note: Multi-peak and irregular domain functions test the agent's adaptability to complex structures.

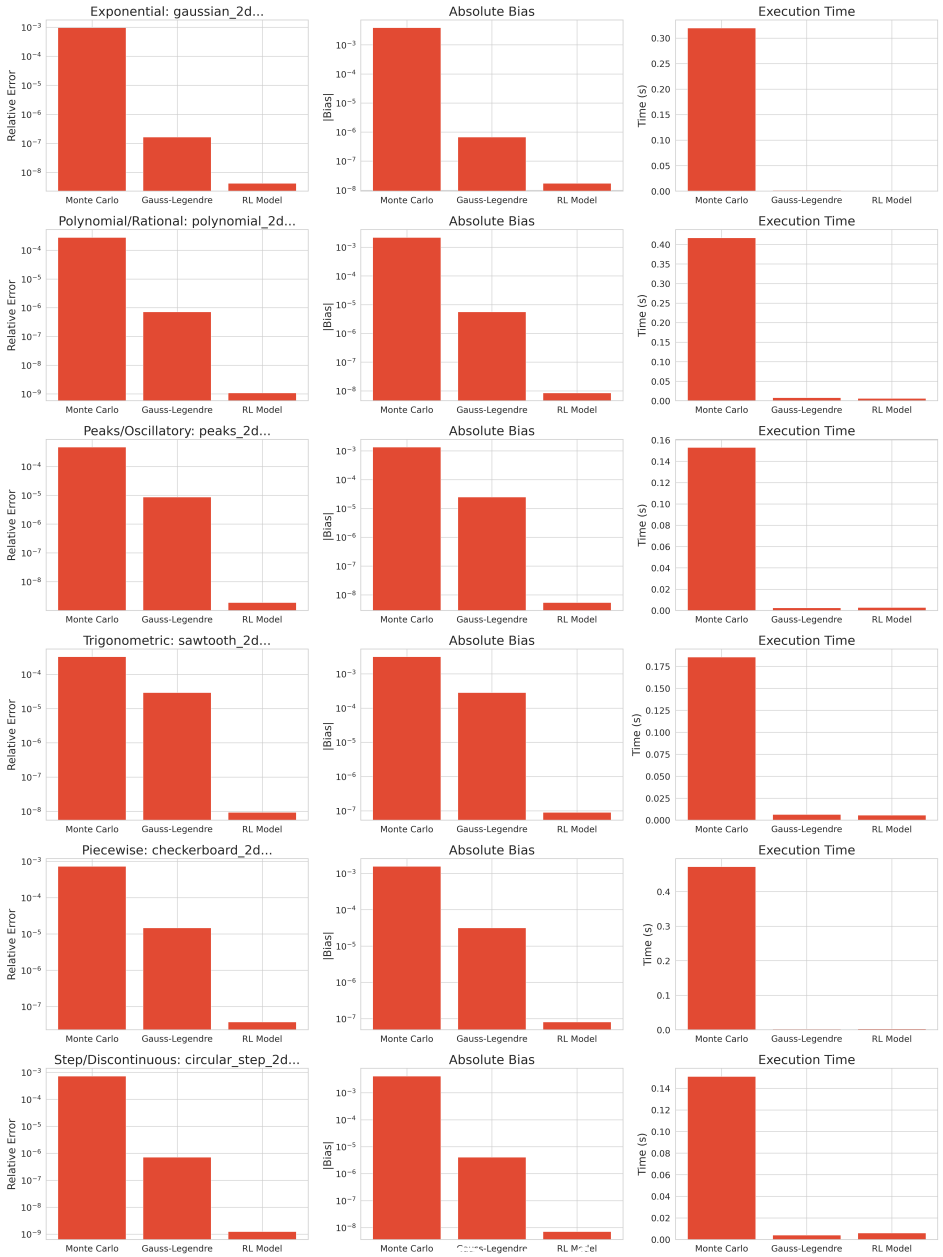
Evaluation Results



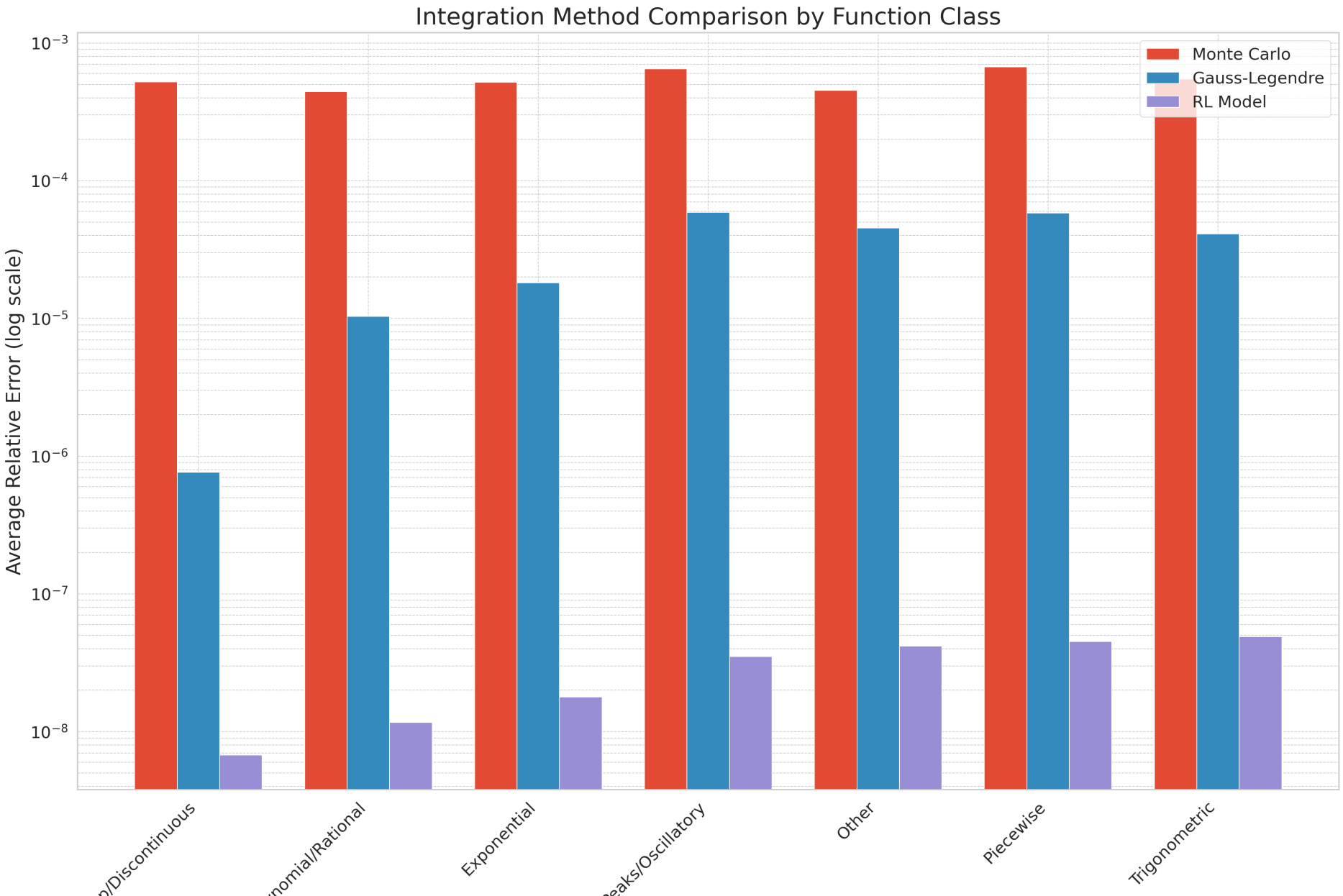
Evaluation Results



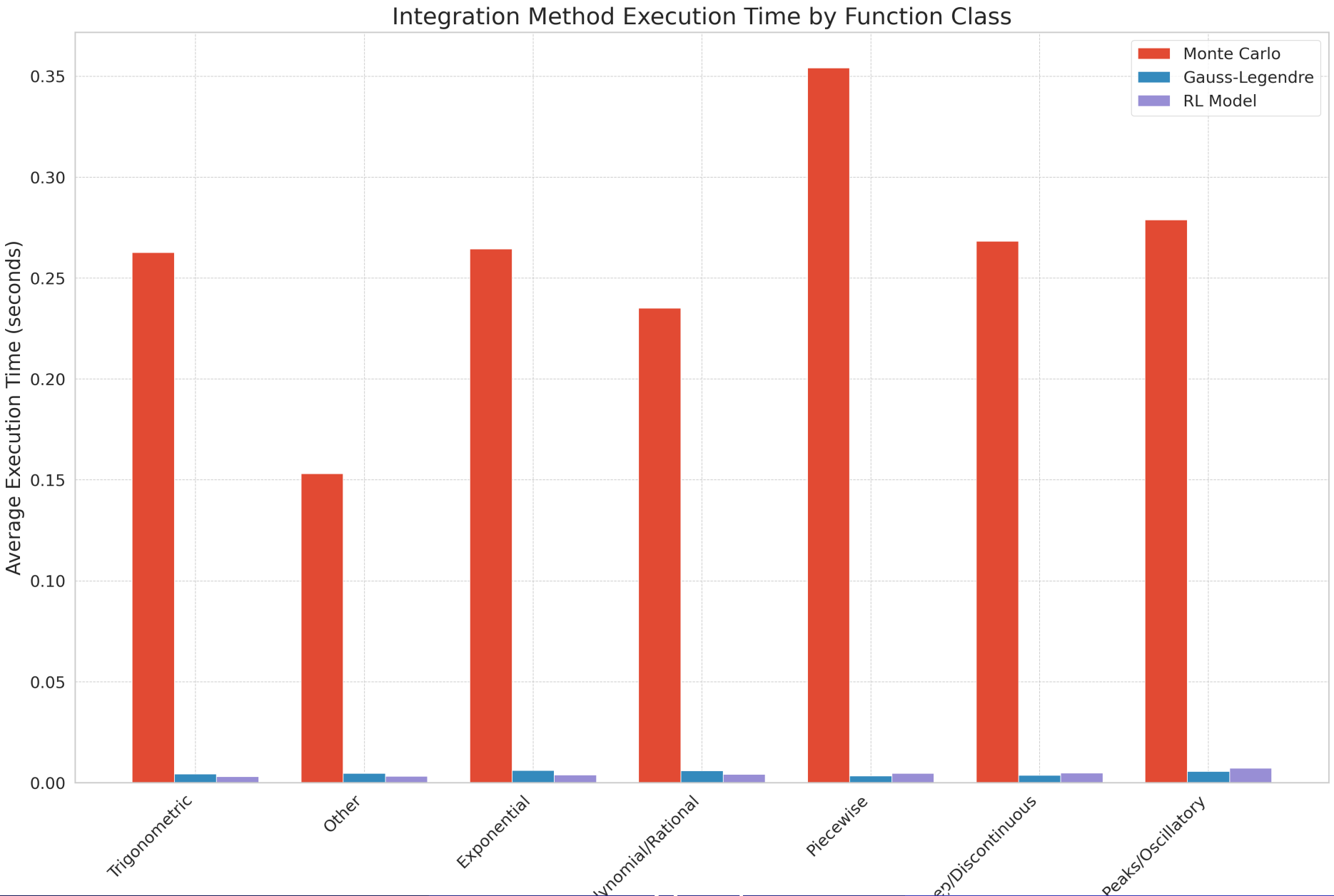
Evaluation Results



Evaluation Results



Evaluation Results



Evaluation Results

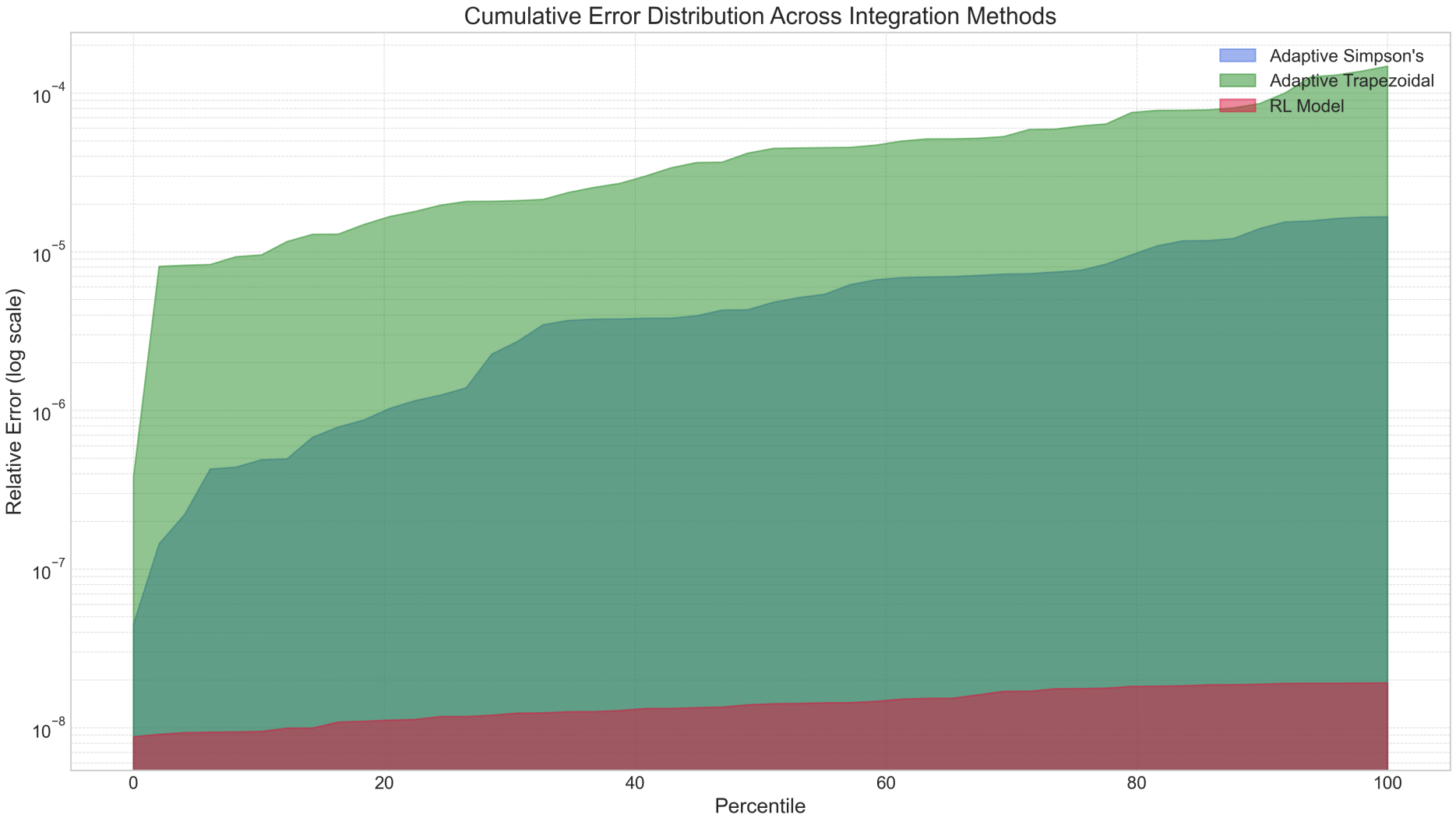


Figure: Grpah of Cummulative error across Method

Evaluation Results

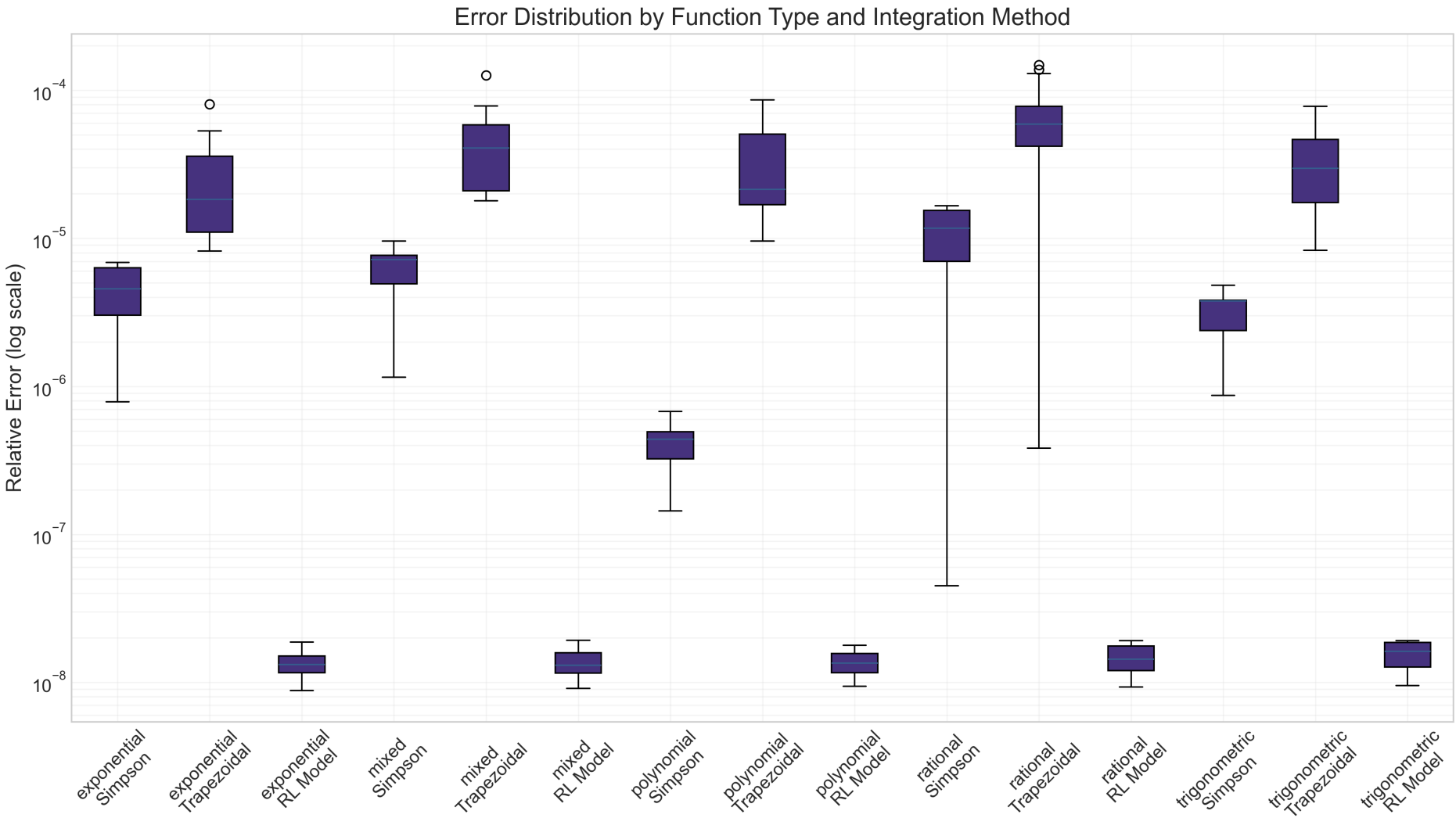


Figure: Box Plot of Error vs Function Class and Method

Evaluation Results

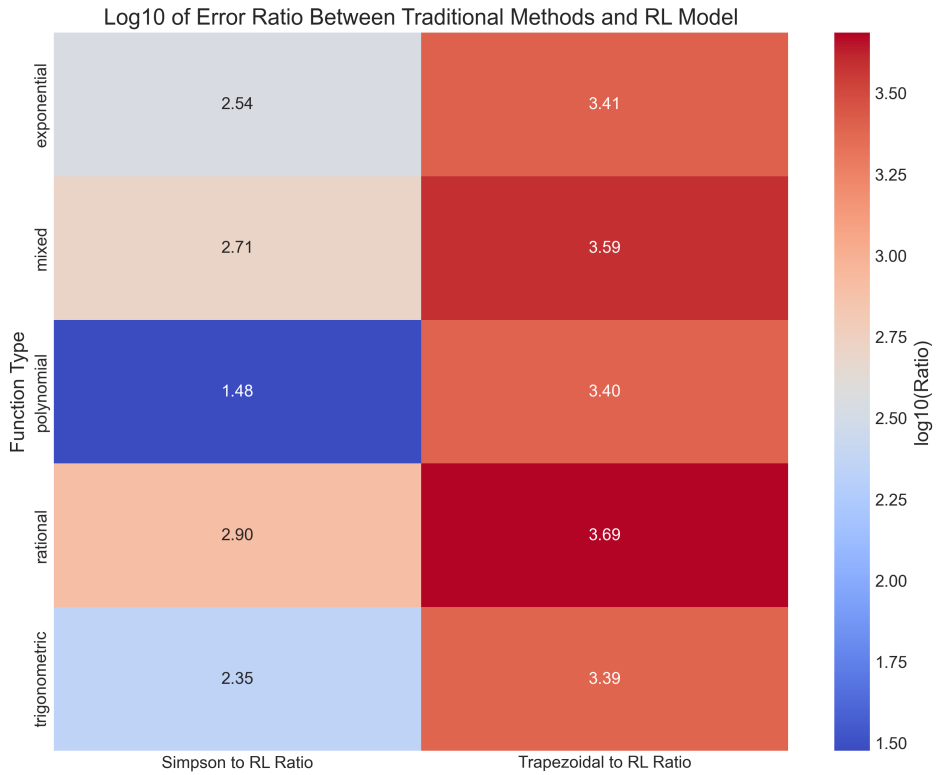
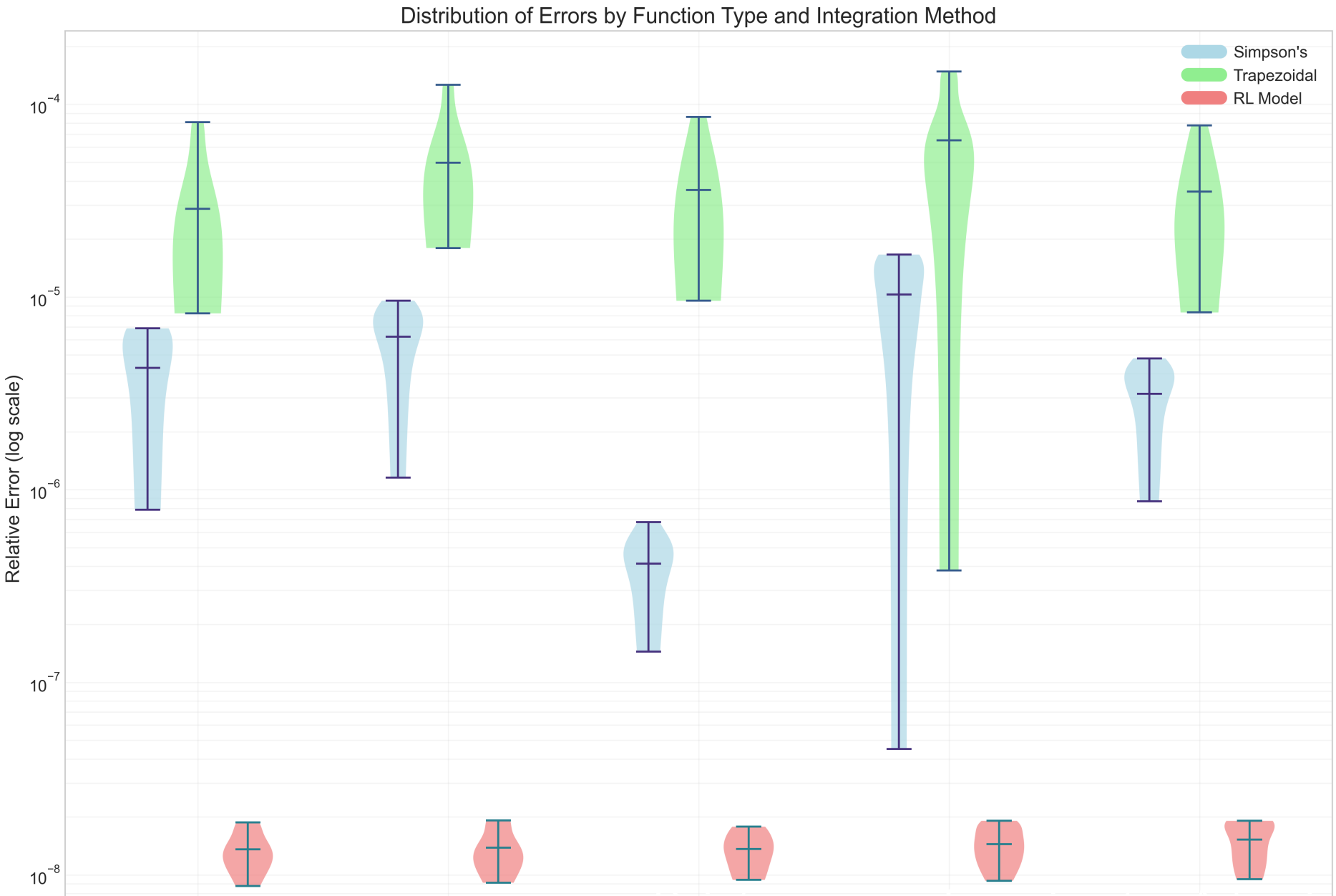
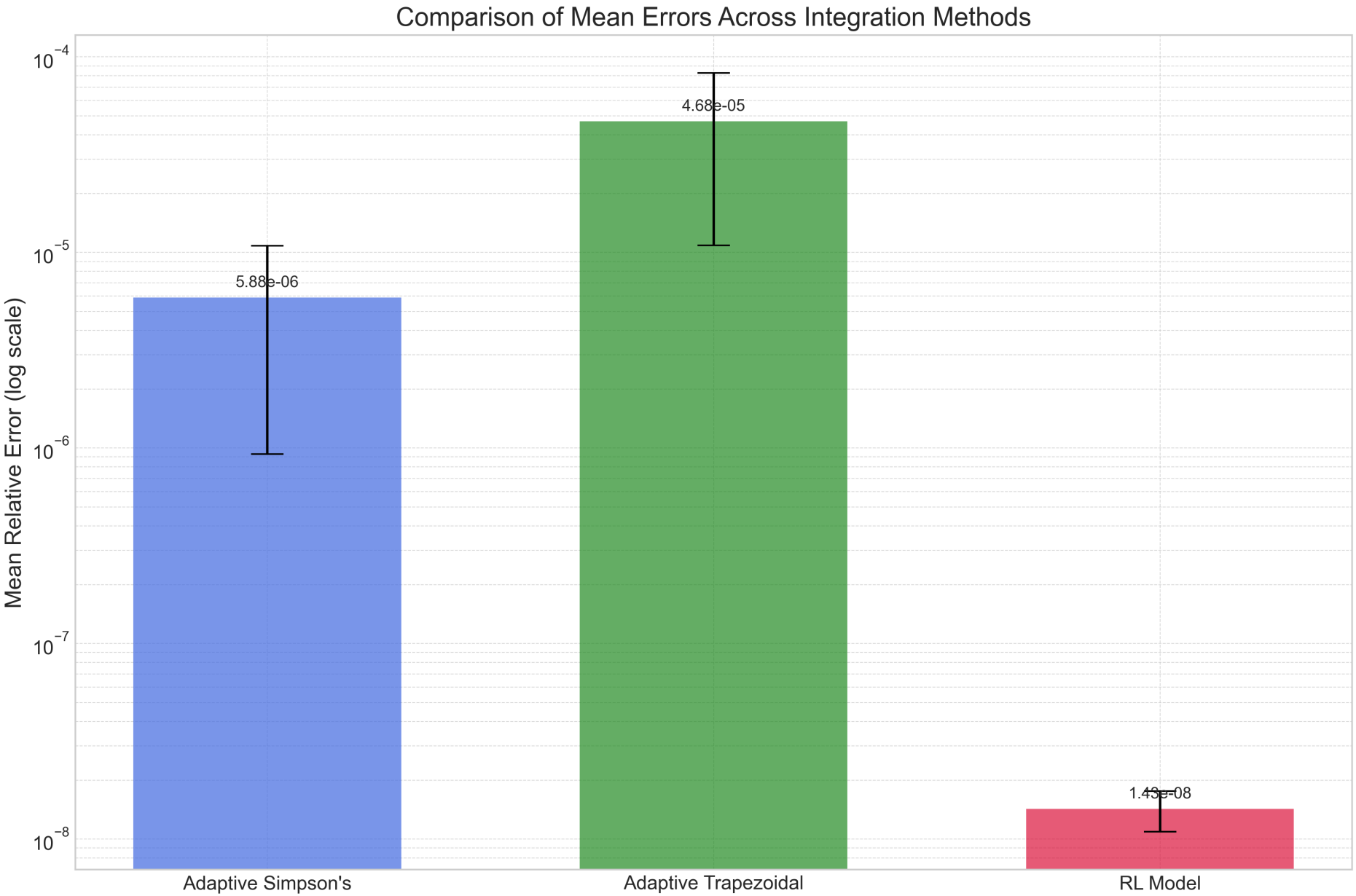


Figure: Comparison Heat Map

Evaluation Results



Evaluation Results



Evaluation Results

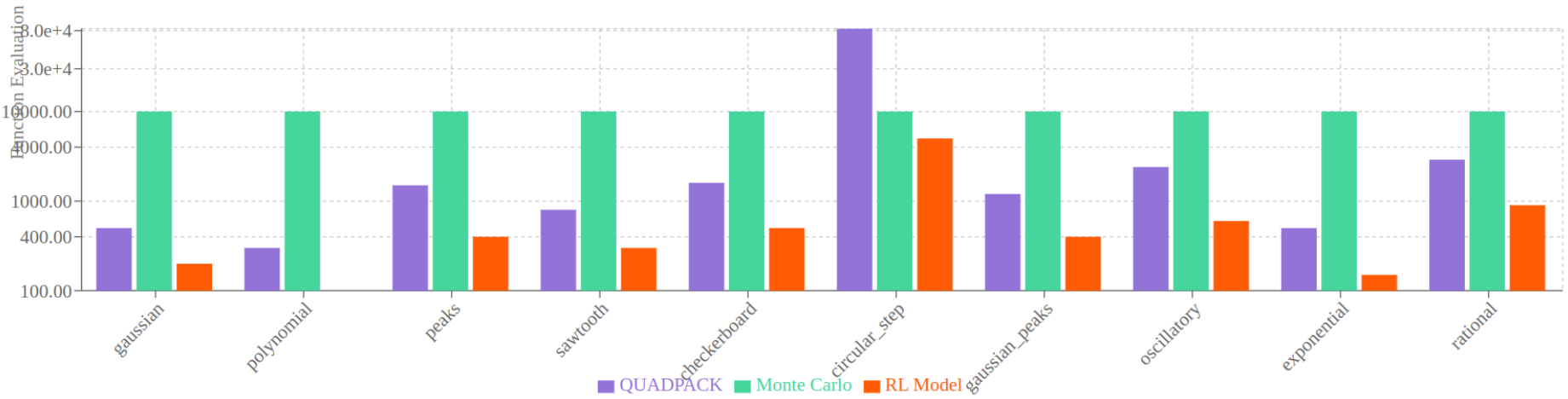


Figure: Time vs Integration Method

Evaluation Results

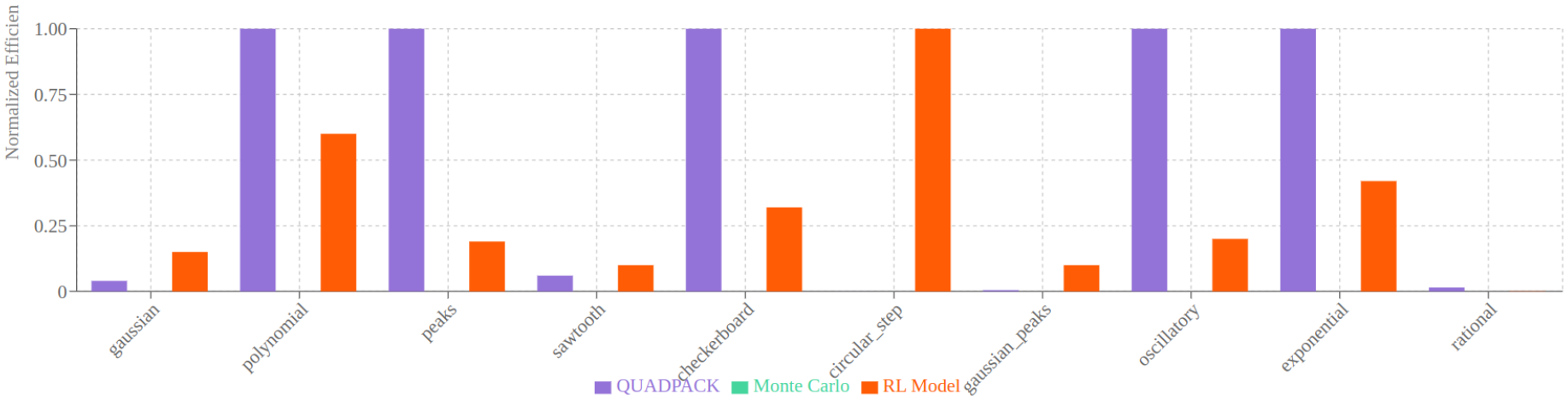


Figure: Efficiency vs Methods

Evaluation Results

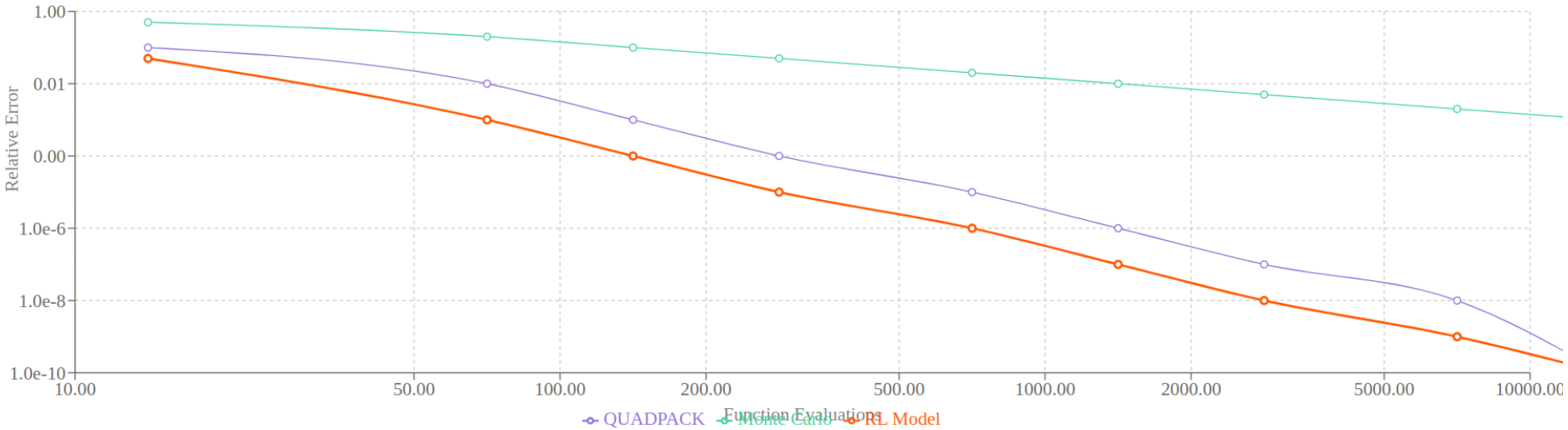
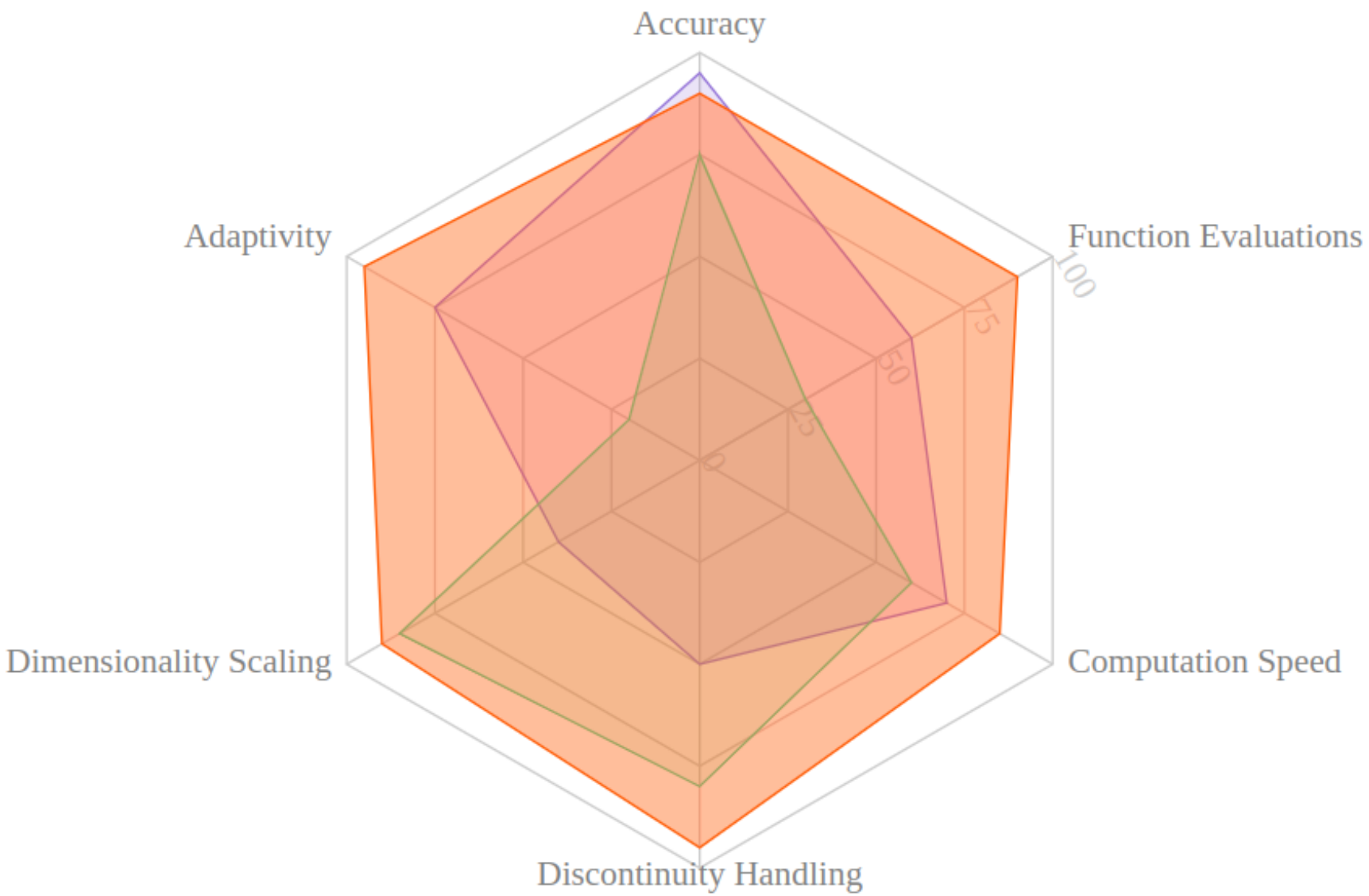


Figure: Relative Error vs Function Evaluations

Evaluation Results



Conclusion

- The implementation of reinforcement learning for adaptive 2D integration effectively allocates computational resources to regions with complex function behavior such as oscillations, discontinuities, and singularities.
- By dynamically selecting regions for refinement based on error estimates, curvature, and local function statistics, the RL agent successfully boosts the accuracy to a large extent.
- The use of multiple integration methods, including Gauss-Legendre Quadrature,Adaptive Simpson, Monte Carlo, and PFEM, enables the model to adapt to various function classes, enhancing overall robustness.

Future Work

- **Optimizing Computational Cost:**
  - Implement advanced pruning strategies to reduce unnecessary function evaluations.
  - Integrate cost-aware reward structures to prioritize high-impact regions.
- **Extending to N-Dimensional Functions:**
  - Develop scalable RL architectures capable of handling higher-dimensional integration.
  - Maintain accuracy across increasing dimensionality while managing computational complexity.

Thank You!

Questions?