ANN-1D-PIB

Artificial Neural Network Solution for the 1D Schrödinger Equation

Nikola Donov

Università degli Studi di Padova

2025 / 2026

Introduction

The Problem

Time-independent Schrödinger equation:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)$$

The Problem

Time-independent Schrödinger equation:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)$$

Problems:

- Analytical solutions only for simple systems
- Complex potentials require numerical methods (FEM, finite difference)

Machine learning (Neural Networks) as a replacement?

The Problem

Time-independent Schrödinger equation:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)$$

Problems:

- Analytical solutions only for simple systems
- Complex potentials require numerical methods (FEM, finite difference)

Machine learning (Neural Networks) as a replacement?

Goal: Train an ANN to predict energy eigenvalues for the 1D particle in a box with polynomial potentials

State of the Art

Recent ML approaches to quantum systems:

- Mills et al. (2017): Deep NNs for 2D Schrödinger equations
- Han et al. (2019): Many-electron systems (He, H_2, Be, B, LiH, H_{10})
- PINNs (Raissi et al., 2019): Encode physics laws in loss function

State of the Art

Recent ML approaches to quantum systems:

- Mills et al. (2017): Deep NNs for 2D Schrödinger equations
- Han et al. (2019): Many-electron systems (He, H_2, Be, B, LiH, H_{10})
- PINNs (Raissi et al., 2019): Encode physics laws in loss function

Simpler ML methods in chemistry:

- Random Forest & Kernel Ridge Regression proven effective
- Training efficiency, interpretability
- Good on tabular data with moderate nonlinearity

Methods

Problem Formulation

Particle in a box (–L to +L):

- Boundary conditions: $\psi(-L) = \psi(+L) = 0$
- Potential inside: $V(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4$

Problem Formulation

Particle in a box (–L to +L):

- Boundary conditions: $\psi(-L) = \psi(+L) = 0$
- Potential inside: $V(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4$

Discretization:

- Discretized into N = 512 grid points
- Finite difference approximation for second derivative
- Transforms PDE into matrix eigenvalue problem: $H\psi = E\psi$

Problem Formulation

Particle in a box (–L to +L):

- Boundary conditions: $\psi(-L) = \psi(+L) = 0$
- Potential inside: $V(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4$

Discretization:

- Discretized into N = 512 grid points
- Finite difference approximation for second derivative
- Transforms PDE into matrix eigenvalue problem: $H\psi = E\psi$

Analytical validation (V = 0):

$$E_n = \frac{n^2 \pi^2 \hbar^2}{2mL^2}, \quad n = 1, 2, 3, \ldots$$

Fortran Numerical Solver

Implementation details:

- Subroutines contained within a separate module
- Polynomial evaluation using Horner's method
- LAPACK's DSYEV routine for eigenvalue computation

Fortran Numerical Solver

Implementation details:

- Subroutines contained within a separate module
- Polynomial evaluation using Horner's method
- LAPACK's DSYEV routine for eigenvalue computation

Validation results:

- Error < 0.001% for all eigenvalues when V(x) = 0
- Double precision (13 significant figures)
- Good numerical results to train our neural network on

Validation

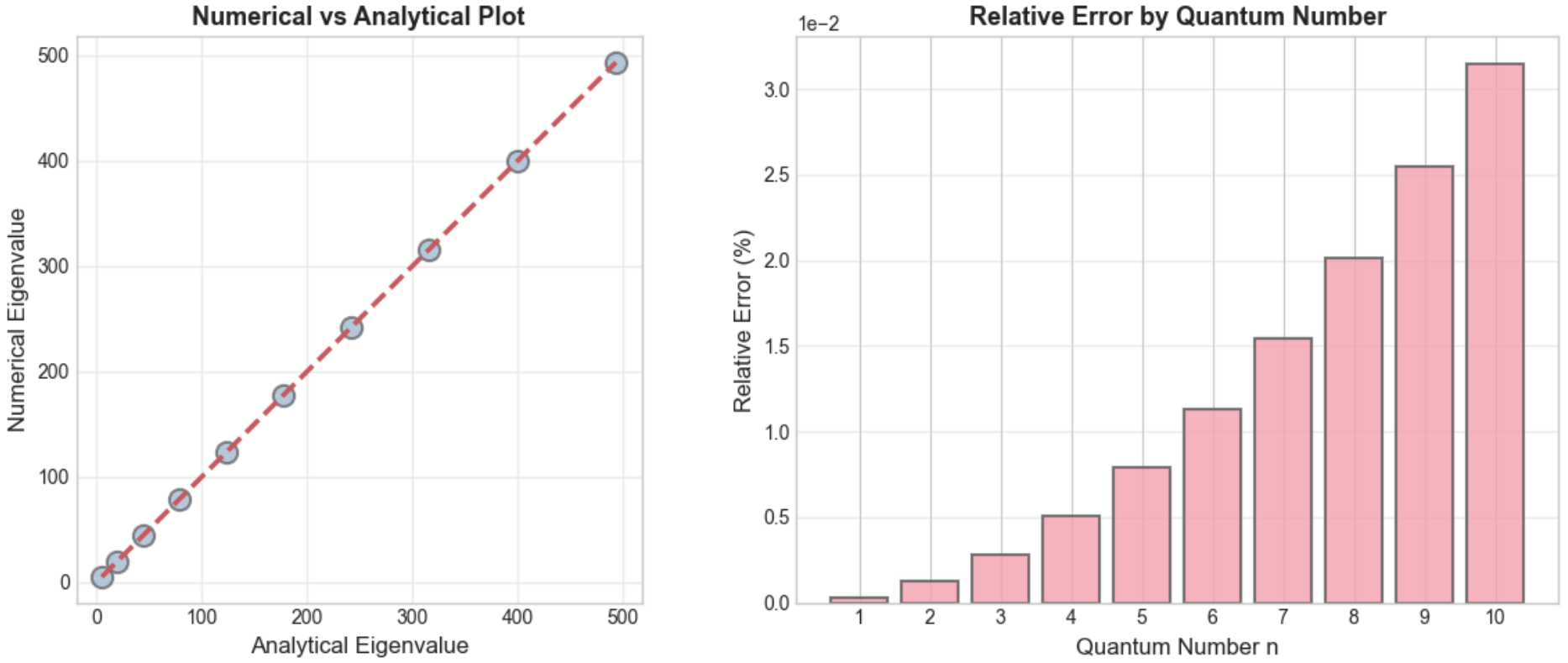

Figure 1: Validation of the numerical solver against the analytical solutions for $V(x) = 0$.

Fortran Code Highlights

Reading input with dynamic allocation:

```
DO
  READ(UNIT = 11, FMT = *, IOSTAT = ios)
  IF (ios /= 0) EXIT
  n = n + 1
END DO
ALLOCATE(coefficients(n))
```

Horner's method for polynomial evaluation:

```
build_poly = coefficients(1)
DO i = 2, SIZE(coefficients)
  build_poly = build_poly * x + coefficients(i)
END DO
```

Fortran Code Highlights

Building Hamiltonian matrix:

```
H(i,i) = 1.0_dp / (dx**2) + V(i)          ! diagonal
H(i, i+1) = -0.5_dp / (dx**2)              ! off-diagonal

CALL DSYEV('V', 'U', N, eigenvectors, N, eigenvalues, WORK,
LWORK, INFO)
```

Dataset Generation

Python wrapper:

```
t = np.linspace(0.0, 1.0, N, endpoint = True)
a0 = -A + 2 * A * t
a1 = -A + 2 * A * t # ... (a2, a3, a4)

for i in range(N):
    coeffs = [a0[i], a1[i], a2[i], a3[i], a4[i]]
    with open('coefficients.txt', 'w') as f:
        f.write(f"{L}\n")
        for coeff in coeffs:
            f.write(f"{coeff}\n")
    subprocess.run([exe_path], cwd = dataset_dir, check =
True)
```

Dataset Generation

Dataset Generator Contains:

- Fixed box length: L = 1.0 ($E \propto \frac{1}{L^2}$)
- Coefficient range: A = 10.0
- Default samples: N = 1024

Dataset Generation

Dataset Generator Contains:

- Fixed box length: L = 1.0 ($E \propto \frac{1}{L^2}$)
- Coefficient range: A = 10.0
- Default samples: N = 1024

Limitation:

- All coefficients share same value per sample: $c_0 = c_1 = c_2 = c_3 = c_4$
- Reduces polynomial diversity
- Performs better than independent random coefficients!

Dataset Generation

Dataset Generator Contains:

- Fixed box length: L = 1.0 ($E \propto \frac{1}{L^2}$)
- Coefficient range: A = 10.0
- Default samples: N = 1024

Limitation:

- All coefficients share same value per sample: $c_0 = c_1 = c_2 = c_3 = c_4$
- Reduces polynomial diversity
- Performs better than independent random coefficients!

Output: Each row contains L, 5 coefficients, 10 eigenvalues

Neural Network Architecture

Multi-Layer Perceptron (MLP):

- Input layer: 6 neurons (L + 5 coefficients)
- Hidden layers: (256, 128, 64) - progressively narrowing
- Output layer: 5 neurons (first 5 eigenvalues)

Neural Network Architecture

Multi-Layer Perceptron (MLP):

- Input layer: 6 neurons (L + 5 coefficients)
- Hidden layers: (256, 128, 64) - progressively narrowing
- Output layer: 5 neurons (first 5 eigenvalues)

Training configuration:

- Activation: `ReLU` (significantly outperformed tanh)
- Optimizer: `Adam` (learning rate $\alpha = 0.001$)
- Loss: Mean Squared Error
- L2 regularization: $\lambda = 0.01$
- Max iterations: 1024

Benchmark Model

Random Forest Regressor:

- Ensemble of 300 decision trees
- Each tree trained on random subset (bagging)
- Prediction: average across all trees

Benchmark Model

Random Forest Regressor:

- Ensemble of 300 decision trees
- Each tree trained on random subset (bagging)
- Prediction: average across all trees

Why Random Forest?

- Proven effective on tabular data
- Good baseline for comparison
- Less prone to overfitting on small datasets

Results

Model Performance

Overall metrics (N = 1024, predicting 5 eigenvalues):

Model	R^2	MSE
Random Forest	0.9999959	0.000152
Neural Network	0.9998474	0.005777

Model Performance

Overall metrics (N = 1024, predicting 5 eigenvalues):

Model	R^2	MSE
Random Forest	0.9999959	0.000152
Neural Network	0.9998474	0.005777

Surprising result: Random Forest outperforms Neural Network!

- 38× lower MSE despite far fewer parameters
- Both achieve excellent R^2 scores (> 0.999)

Predicted vs. Numerical Eigenvalues

Neural Network predictions:

- Near-perfect comparison with numerical solver
- No significant difference between lower and higher eigenvalues
- Stable up to 4th excited state

Predicted vs. Numerical Eigenvalues

Neural Network predictions:

- Near-perfect comparison with numerical solver
- No significant difference between lower and higher eigenvalues
- Stable up to 4th excited state

Error analysis by quantum number:

- Absolute error increases with quantum number
- Relative error **decreases** for higher eigenvalues
- Model actually performs better on higher energy states in relative terms

Predicted vs. Numerical Eigenvalues Graphs

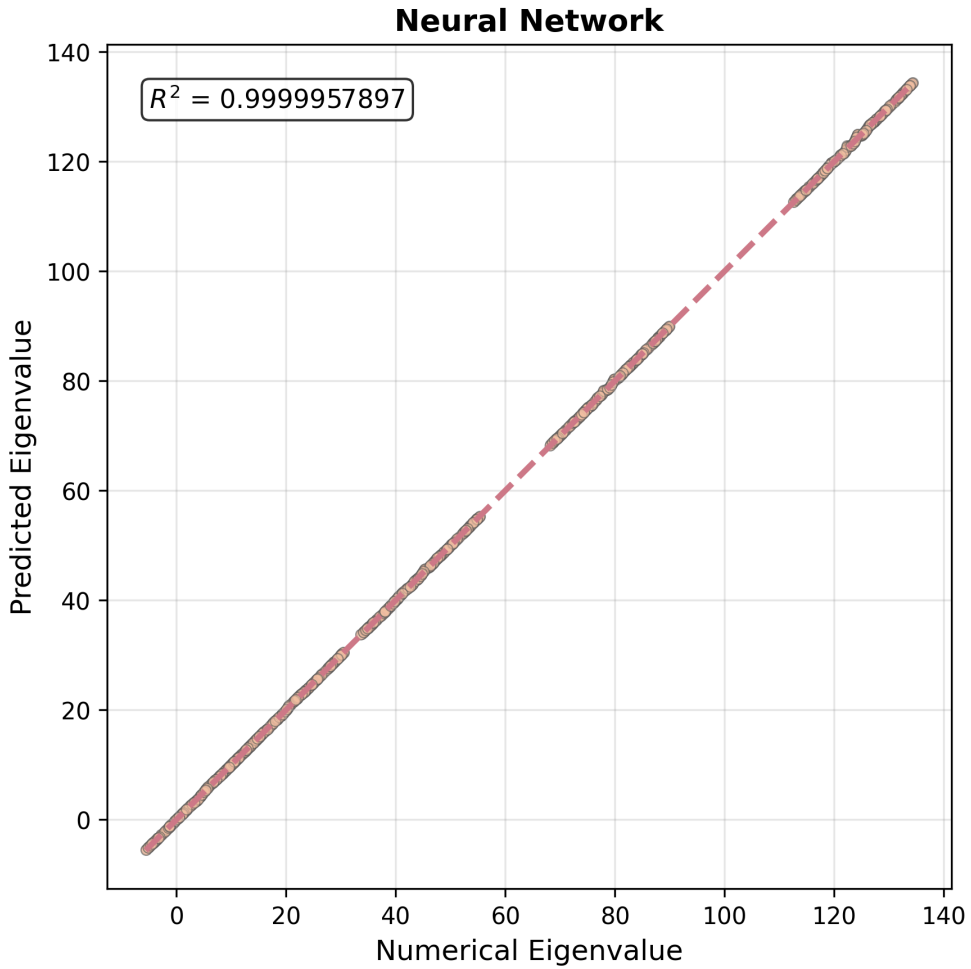

Figure 2: Predicted vs. Numerical Eigenvalues of the system

Predicted vs. Numerical Eigenvalues Graphs

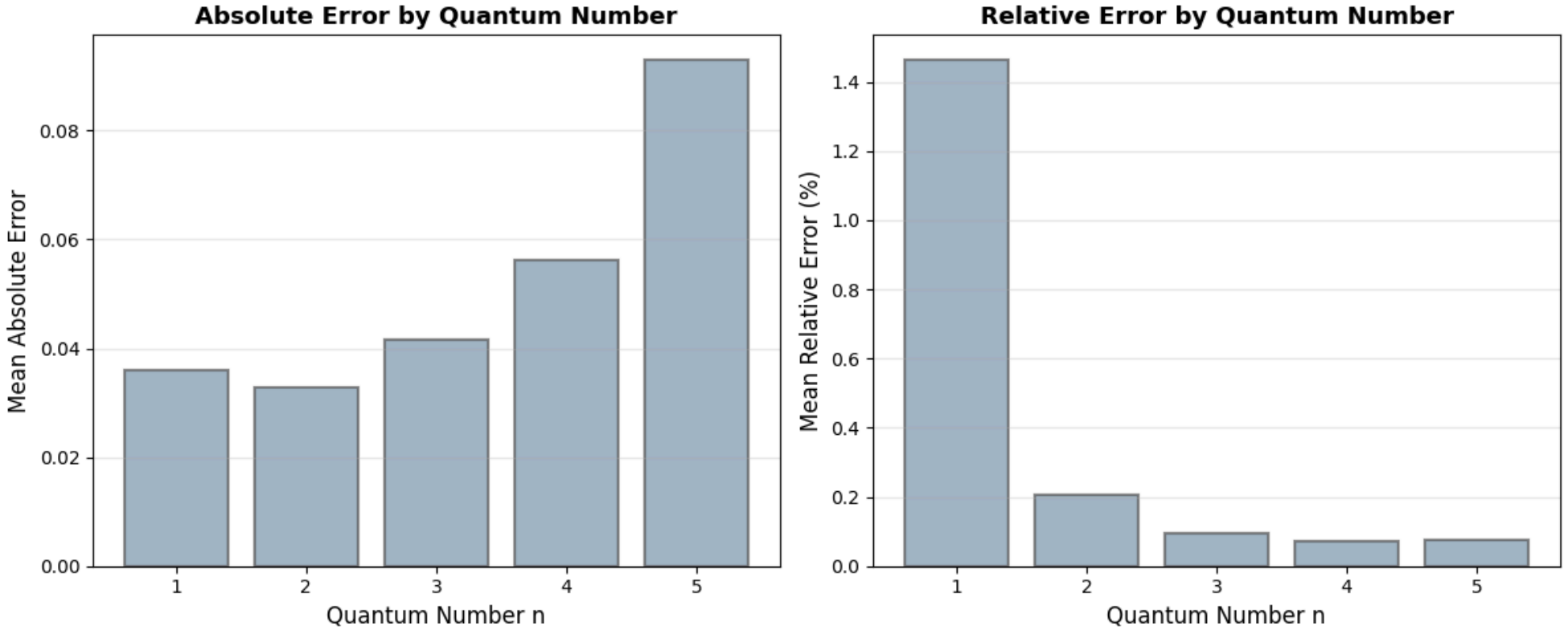

Figure 3: Error distribution per quantum number in the neural network Model.

Parameter Sensitivity: Dataset Size

Effect of N on Neural Network performance:

N	R^2	MSE
512	0.9998113	0.007847
1024	0.9998474	0.005777
2048	0.9998837	0.004543
4096	0.9999191	0.003270

Parameter Sensitivity: Dataset Size

Effect of N on Neural Network performance:

N	R^2	MSE
512	0.9998113	0.007847
1024	0.9998474	0.005777
2048	0.9998837	0.004543
4096	0.9999191	0.003270

Observations:

- Doubling data: 21.36% MSE improvement
- Quadrupling data: 43.40% MSE improvement
- More data $\rightarrow$ better performance

Parameter Sensitivity: Dataset Size Graph

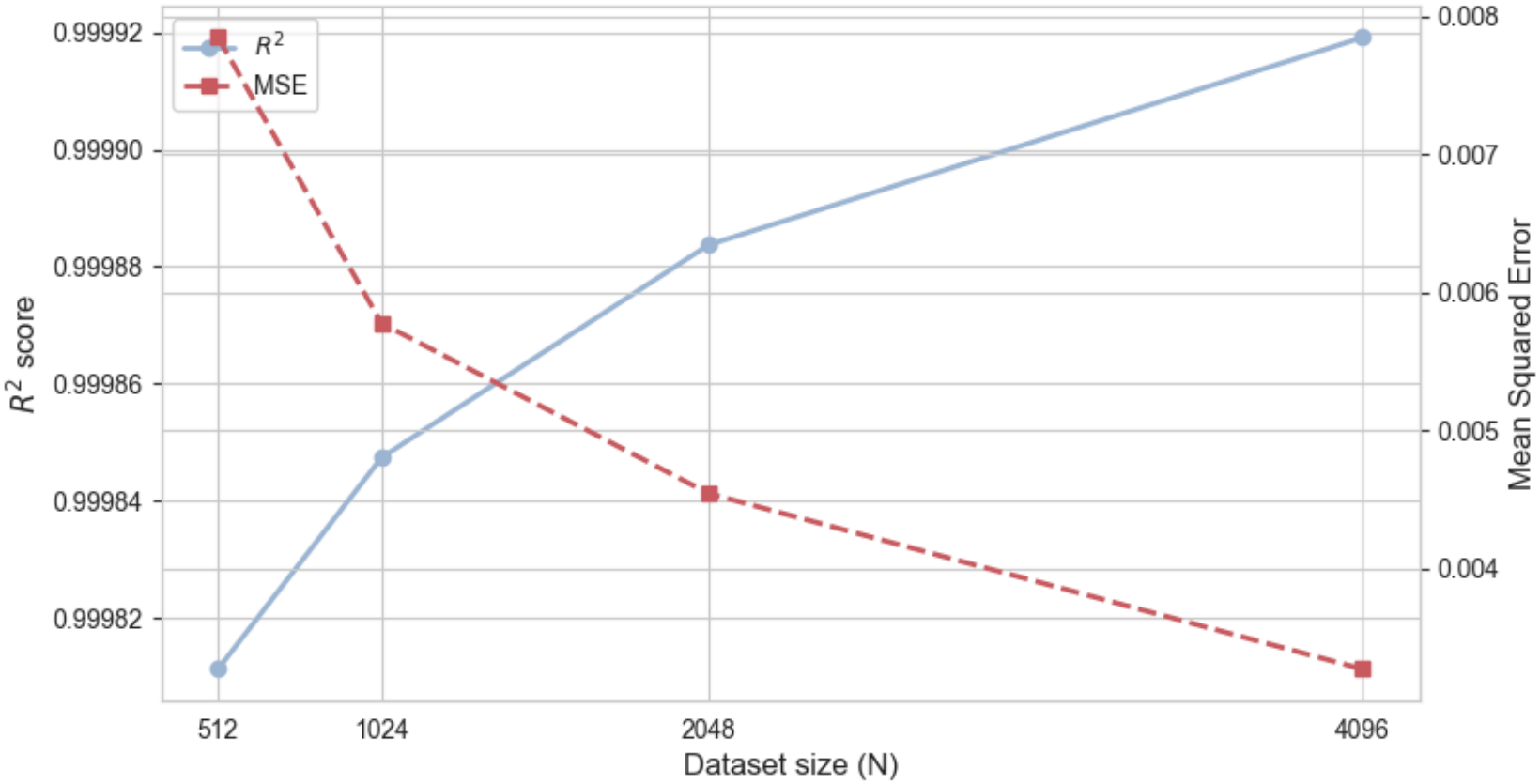

Figure 4: Dependence of the neural network's performance on the size

Parameter Sensitivity: Eigenvalue Count

Predicting 5 vs 10 eigenvalues:

Eigenvalues	R^2	MSE
5	0.9998474	0.005777
10	0.9978428	0.080409

Parameter Sensitivity: Eigenvalue Count

Predicting 5 vs 10 eigenvalues:

Eigenvalues	R^2	MSE
5	0.9998474	0.005777
10	0.9978428	0.080409

Dramatic degradation for higher eigenvalues:

- 1291% increase in MSE
- R^2 drops by factor of 2.5
- Relationship becomes more nonlinear for excited states

Parameter Sensitivity: Coefficient Range

Effect of A on performance:

A	R^2	MSE
5	0.999646	0.003327
10	0.9998474	0.005777
20	0.9999576	0.006351

Parameter Sensitivity: Coefficient Range

Effect of A on performance:

A	R^2	MSE
5	0.999646	0.003327
10	0.9998474	0.005777
20	0.9999576	0.006351

Observations:

- Narrower range (A = 5): 42% lower MSE, but fewer shapes explored
- Wider range (A = 20): Only 10% MSE increase

Random Forest Comparison

Literature:

- Fernández-Delgado et al. (2014): RF achieved top performance 94.1% of time on 121 datasets
- Particularly effective on tabular data with moderate nonlinearity

Our problem characteristics favoring RF:

- Tabular data structure
- Moderate dataset size ($\leq$ 4096 samples)
- Reduced complexity from uniform coefficient scaling

When would NNs excel?

- Larger, more diverse datasets, independent coefficient variation, higher dimensional problems, more complex nonlinear relationships

Conclusions

Summary

- Artificial neural network successfully approximates 1D Schrödinger equation solutions
- Python wrapper integrating Fortran solver with ML models
- High accuracy: $R^2 > 0.9998$ for first 5 eigenvalues

Summary

- Artificial neural network successfully approximates 1D Schrödinger equation solutions
- Python wrapper integrating Fortran solver with ML models
- High accuracy: $R^2 > 0.9998$ for first 5 eigenvalues

Considerations:

- Random Forest outperformed Neural Network despite fewer parameters
- Simple structure favors simpler statistical models

Limitations & Future Work

Current limitations:

- Uniform coefficient scaling reduces diversity: $c_0 = c_1 = c_2 = c_3 = c_4$
- Fixed box length L = 1.0
- Limited to 1D, polynomial potentials
- Performance degrades for higher eigenvalues

Limitations & Future Work

Current limitations:

- Uniform coefficient scaling reduces diversity: $c_0 = c_1 = c_2 = c_3 = c_4$
- Fixed box length L = 1.0
- Limited to 1D, polynomial potentials
- Performance degrades for higher eigenvalues

Future directions:

- Independent coefficient variation for greater diversity
- Variable box length to learn $E \propto \frac{1}{L^2}$ relationship
- Extension to 2D/3D systems
- Physics-Informed Neural Networks (PINNs) approach
- Larger, more diverse datasets
- More complex potential functions

Final Thoughts

Conclusion:

Machine learning approaches can serve as alternatives or complements to traditional numerical methods for quantum problems

Final Thoughts

Conclusion:

Machine learning approaches can serve as alternatives or complements to traditional numerical methods for quantum problems

But...

Choice of model depends on:

- Problem structure and complexity
- Dataset characteristics
- Required accuracy vs. computational cost

Final Thoughts

Conclusion:

Machine learning approaches can serve as alternatives or complements to traditional numerical methods for quantum problems

But...

Choice of model depends on:

- Problem structure and complexity
- Dataset characteristics
- Required accuracy vs. computational cost

Thank you for your time!