Neural Network-Based PDE Solver Using the Feynman-Kac Formula

Shubha Sanket Samantaray

Université Grenoble Alpes

November 27, 2025

Motivation

- Partial Differential Equations (PDEs) are fundamental tools for modeling dynamic systems in physics, biology, finance, and engineering.
- Classical solvers such as finite differences and finite elements are effective in low dimensions but become computationally intractable in high-dimensional settings due to the curse of dimensionality.
- Find an alternate method to solve PDEs in a mesh-free setting, being able to perform well in higher dimensions as well.
- Feynman–Kac formula reformulates certain linear parabolic PDEs as expectations over stochastic processes, from deterministic solvers to statistical estimation.

The Heat Equation: A Foundational Example

- **Backward form (terminal condition):**

$$\partial_t u + \frac{1}{2}\Delta u = 0, \quad u(T, x) = f(x)$$

- **Feynman–Kac solution:**

$$u(t, x) = \mathbb{E}\left[f(W_T) \mid W_t = x\right]$$

- **Forward form (initial condition):**

$$\partial_t v - \frac{1}{2}\Delta v = 0, \quad v(0, x) = f(x)$$

- **Feynman–Kac solution:**

$$v(t, x) = \mathbb{E}^x\left[f(W_t)\right] = \mathbb{E}\left[f(W_t) \mid W_0 = x\right]$$

- W_t denotes a standard Brownian motion in $\mathbb{R}^d$.

Feynman–Kac for Linear Parabolic PDEs

- Consider the PDE:

$$\partial_t u + b(x)\partial_x u + \frac{1}{2}\sigma^2(x)\partial_{xx} u - ru = 0, \quad u(T,x) = f(x)$$

- **Feynman–Kac representation:**

$$u(t,x) = \mathbb{E}\left[e^{-r(T-t)} f(X_T) \mid X_t = x\right]$$

- **Where** X_s satisfies the SDE:

$$dX_s = b(X_s)\,ds + \sigma(X_s)\,dW_s$$

- Connects PDE solutions to stochastic processes through conditional expectations.

The Black–Scholes Model

- The Black–Scholes model provides a framework for pricing European options.
- A European call option gives its holder, the right (but not the obligation) to buy an asset at strike price K at maturity T.
- The option price $V(t,S)$ satisfies the celebrated Black–Scholes PDE:

$$\partial_t V + rS\,\partial_S V + \frac{1}{2}\sigma^2 S^2\,\partial_{SS} V - rV = 0,$$

  with terminal condition:

$$V(T,S) = \Phi(S) = \max(S-K,0).$$

- This PDE models the evolution of option prices under a no-arbitrage assumption and constant volatility.

Feynman–Kac Representation of Black–Scholes

- The Black–Scholes PDE is a special case of a linear parabolic PDE:

$$\partial_t u + b(x)\partial_x u + \frac{1}{2}\sigma^2(x)\partial_{xx} u - ru = 0$$

  with:

$$x = S, \quad b(S) = rS, \quad \sigma(S) = \sigma S.$$

- Under the risk-neutral measure $\mathbb{Q}$, the asset follows:

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t^{\mathbb{Q}}.$$

- Feynman–Kac gives the option price as:

$$V(t,S) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}\Phi(S_T) \mid S_t = S\right].$$

- In particular, for a European call:

$$C(t,S) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}(S_T - K)^+ \mid S_t = S\right].$$

Neural Networks as Conditional Expectation Approximators

- The Feynman–Kac formula expresses the PDE solution as a conditional expectation:

$$u(t, x) = \mathbb{E}\left[e^{-r(T-t)} f(X_T) \mid X_t = x\right]$$

- This is the solution to the regression problem, with the MSE loss:

$$\varphi^\star = \arg\min_{\varphi} \mathbb{E}\left[(g(X_T) - \varphi(X_t))^2\right]$$

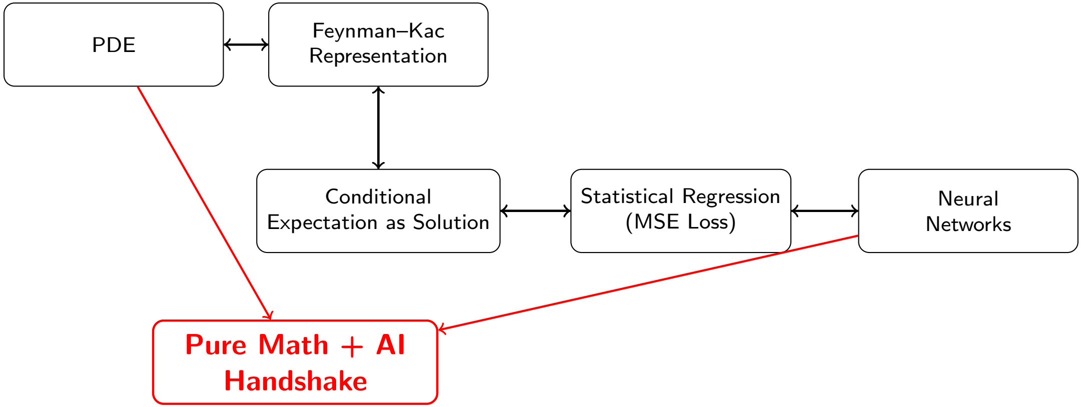
- The unique minimizer is the conditional expectation:

$$\varphi^\star(x) = \mathbb{E}\left[g(X_T) \mid X_t = x\right]$$

- Note that in this setup, we have to train the model by keeping the time t fixed.

PDEs, Probabilities, and Neural Networks: A Conceptual Map

A conceptual flow: from mathematical theory to data-driven approximation.


PDE
Feynman–Kac Representation
Conditional Expectation as Solution
Statistical Regression (MSE Loss)
Neural Networks
Pure Math + AI Handshake

Learning Procedure

1. **Sample Generation:** Simulate trajectories of the underlying SDE to obtain pairs (X_t^i, X_T^i).
2. **Label Construction:** Evaluate the terminal payoff to compute training targets: $y^i = f(X_T^i)$.
3. **Model Training:** Fit a neural network φ_θ by minimizing the mean squared error between $\varphi_\theta(X_t^i)$ and y^i.
4. **Solution Recovery:** Use the trained model $\varphi_\theta(x) \approx \mathbb{E}[f(X_T) \mid X_t = x]$ as an approximation of the PDE solution.

Data Generation for European Call Option

Two Sampling Strategies:

- **Closed-form (GBM):**

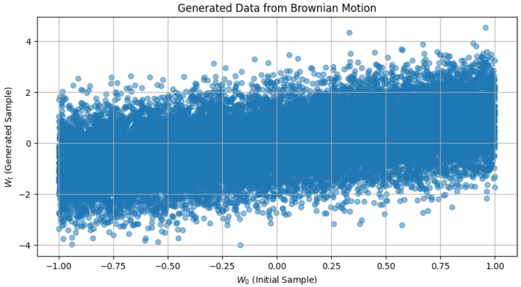
$$S_T = S_t \exp\left((r - \frac{1}{2}\sigma^2)(T - t) + \sigma\sqrt{T - t}Z\right),\ Z \sim \mathcal{N}(0,1)$$

- **Euler–Maruyama discretization:**

$$S_{n+1} = S_n + rS_n\Delta t + \sigma S_n \Delta W_n, \quad \Delta W_n \sim \mathcal{N}(0, \Delta t)$$

Note: This dual-scheme setup lets us assess accuracy and robustness under both exact and approximate SDE sampling.

Results for the Forward Heat Equation



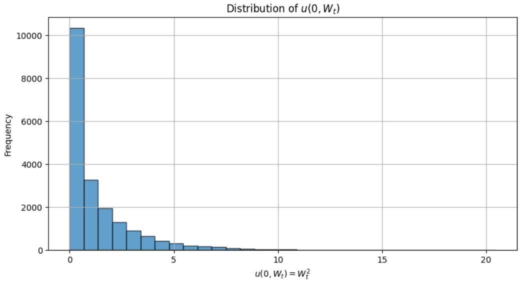
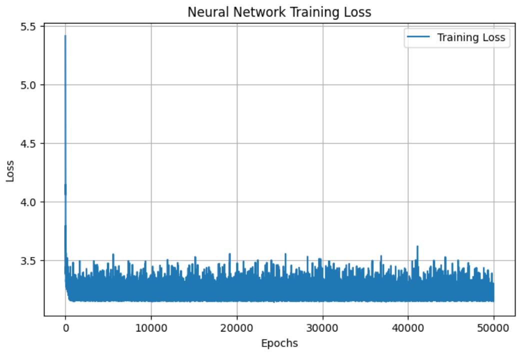


Figure: Left: Scatter plot of Brownian pairs (W_0, W_1). Right: Histogram of target values W_1^2.

Observation: The histogram is positively skewed, consistent with the square of a Gaussian variable.

Heat Equation: Training and Prediction Behavior



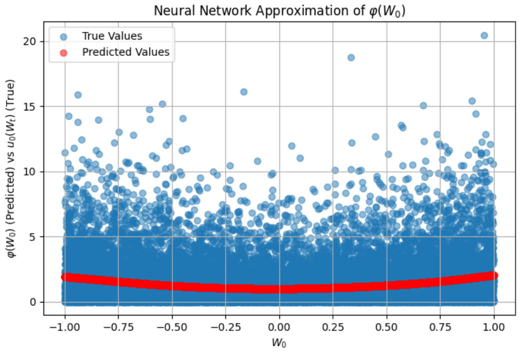
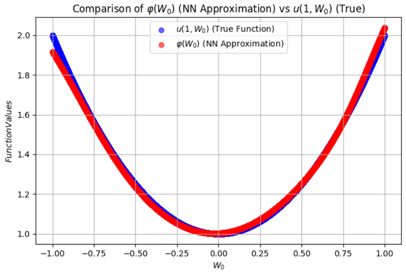


Figure: Left: Training loss curve. Right: Predicted values vs stochastic labels W_1^2.

Observation: Despite label noise, the network learns the conditional mean, not just data interpolation.

Heat Equation: Comparison with Exact Solution

- Overlay shows excellent alignment between predicted and exact solutions.
- The absolute pointwise error is uniformly small, staying below 0.05 throughout the domain.



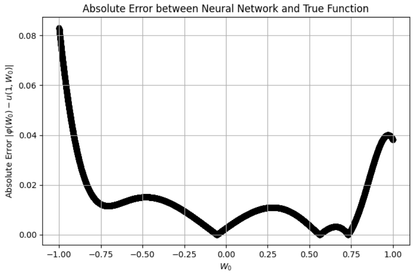
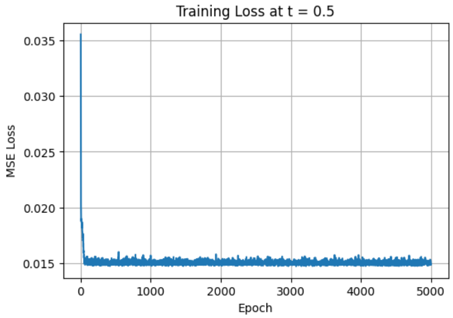


Figure: Left: Predicted function vs exact $x^2 + 1$. Right: Absolute error across domain.

Conclusion: The Feynman–Kac solver accurately recovers the heat equation solution and validates the regression framework.

Black–Scholes: Euler–Maruyama Sampling

- Asset paths simulated via Euler–Maruyama discretization of GBM.
- Network trained for 5000 epochs to learn pricing map at $t = 0.5$.
- Predicted prices show close alignment with the analytical Black–Scholes solution.



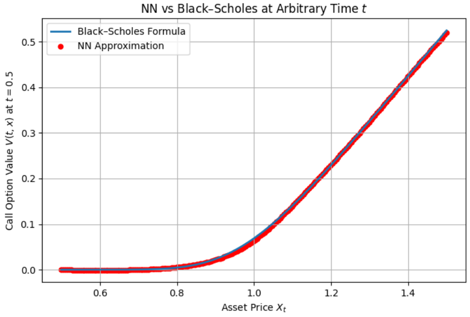
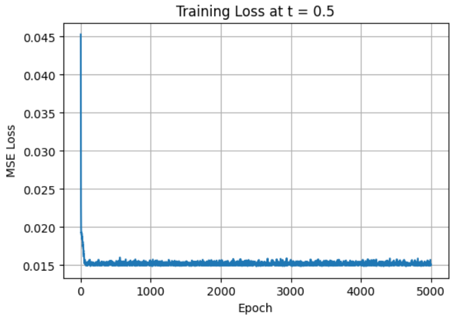


Figure: Left: Training loss convergence. Right: NN predictions vs analytical Black–Scholes prices.

Black–Scholes: Closed-Form Sampling

- Terminal stock prices generated via exact GBM formula.
- Neural network again trained for 5000 epochs.
- Even better approximation observed compared to Euler–Maruyama.



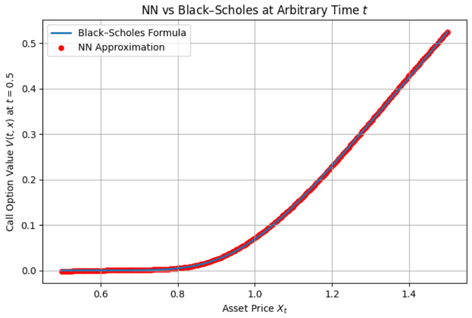


Figure: Left: Training loss for closed-form data. Right: NN predictions vs Black–Scholes values.

Black–Scholes: Discussion and Insights

- Both data generation schemes produce accurate neural approximations of the Black–Scholes price function at fixed time $t = 0.5$.
- **Closed-form simulation** yields lower approximation error:
  - Mean absolute error: 0.001085
  - Maximum error: 0.003068
- **Euler–Maruyama discretization** performs slightly worse:
  - Mean absolute error: 0.001482
  - Maximum error: 0.005502
- The comparison confirms that while both methods validate the neural solver, the exact simulation offers a quantitative edge when available.

Conclusion: Closed-form and numerical schemes both validate the neural Feynman–Kac solver. Euler–Maruyama remains a robust choice when exact simulations are unavailable.

Learning the Full Space-Time Solution Surface

Goal: Extend the fixed-time model $x \mapsto V(t,x)$ to a full surface model

$$(t,x) \mapsto V(t,x) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}\Phi(S_T) \mid S_t = x\right]$$

One network $\varphi_\theta(t,x)$ learns the entire solution surface across $t \in [0, T]$ and $x \in \mathbb{R}_+$.

Motivation:

- In fixed-t experiments, networks learn a slice $x \mapsto V(t,x)$.
- This requires retraining for each new t.
- We aim to extend the input to include both t and x, enabling generalization.

Key Question: Can a neural network learn a map of conditional expectations over time and space, i.e. a continuous family of such expectations?

Training Setup for Space-Time Approximation

Data Generation Procedure:

- Sample $(t^i, x^i) \in [0, T] \times [a, b]$ uniformly or on a grid.
- Simulate terminal value:

$$S_T^i = x^i \cdot \exp\left(\left(r - \frac{1}{2}\sigma^2\right)(T - t^i) + \sigma\sqrt{T - t^i} \cdot Z^i\right), \quad Z^i \sim \mathcal{N}(0, 1)$$

- Compute label:

$$y^i = e^{-r(T - t^i)}\Phi(S_T^i)$$

Learning Objective:

$$\min_\theta \frac{1}{N}\sum_{i=1}^{N}\left(\varphi_\theta(t^i, x^i) - y^i\right)^2$$

Learning the Full Surface

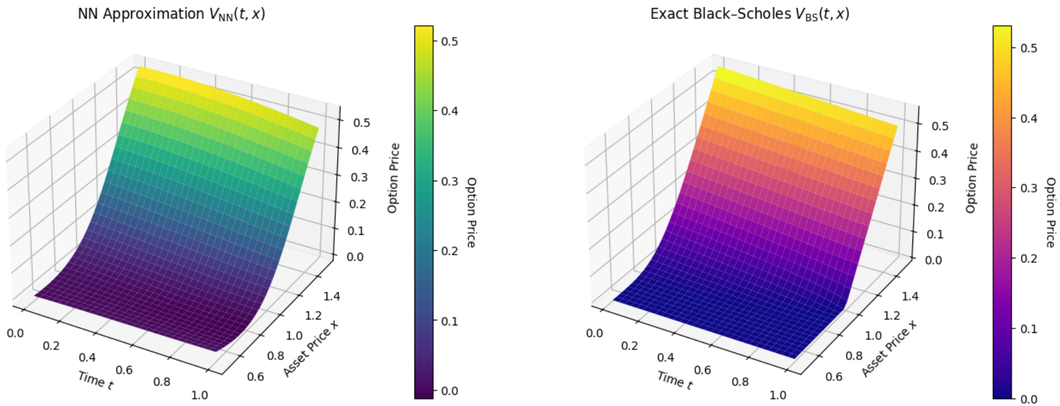
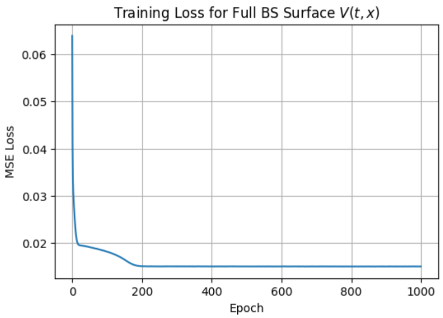


Figure: Left: Neural network surface $V_{\text{NN}}(t, x)$. Right: Exact Black–Scholes solution $V_{\text{BS}}(t, x)$.

Training and Accuracy Evaluation

- Loss stabilizes during training, confirming convergence.
- Learned surface matches analytical solution with low pointwise error.



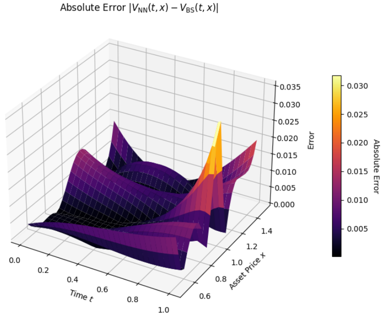
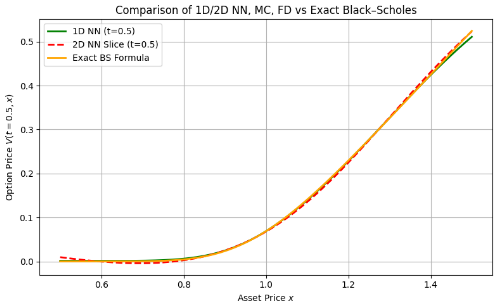


Figure: Left: Training loss over epochs. Right: Absolute error across the full (t, x) domain.

Conclusion: The 2D network successfully generalizes over both time and space.

Comparison with 1D Network and Classical Solvers

- Slice extracted at $t = 0.5$ from the full surface.
- Compares 1D vs 2D networks — both learn the solution accurately.
- All solvers (NNs, MC, FD) show strong agreement with the exact solution.



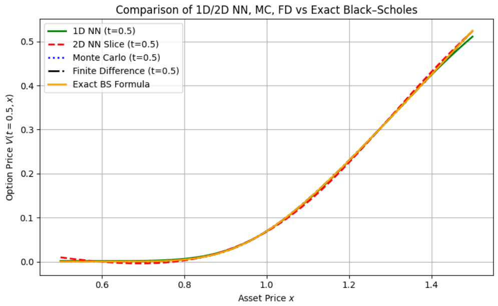
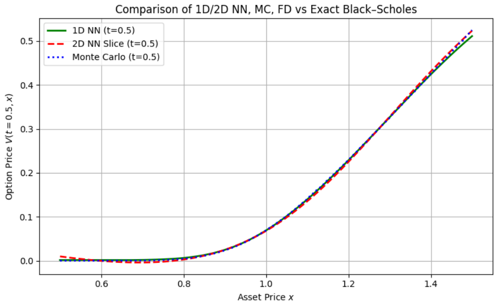


Figure: Left: 1D NN vs 2D NN vs exact at $t = 0.5$. Right: All methods vs exact at $t = 0.5$.

Comparison with 1D Network and Classical Solvers

- Left: 2D network is compared against Monte Carlo simulation at $t = 0.5$.
- Right: Comparison with Crank–Nicolson finite difference scheme.
- Neural network predictions show excellent alignment with reference methods.



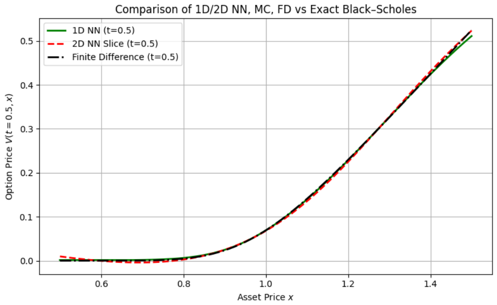


Figure: Left: 1D NN,2D NN vs Monte Carlo. Right: 1D NN, 2D NN vs Finite Difference.

Comparison of Evaluation Methods

- **Monte Carlo:** Computes expectation at each (t, x) via fresh simulations. No reuse across inputs.
- **1D Neural Network:** Learns $x \mapsto V(t, x)$ for fixed t. Requires retraining for each new time.
- **Finite Difference:** Solves on a fixed time-space grid. Interpolation needed for off-grid points.
- **2D Neural Network:** Learns the full map $(t, x) \mapsto V(t, x)$. Predicts values instantly for any input pair.

Key Insight

2D neural networks enable mesh-free, reusable inference across all (t, x) values
— *a significant advantage over simulation and grid-based methods.*

Conclusion

- Reformulated linear parabolic PDEs as conditional expectation problems using the Feynman–Kac formula.
- Trained neural networks on simulated SDE trajectories to approximate the PDE solutions.
- Successfully recovered known solutions to the heat equation and Black–Scholes PDE at fixed time and across full surfaces.
- Achieved strong agreement with finite difference and Monte Carlo methods.
- Enabled mesh-free, data-driven PDE solvers that scale to irregular or high-dimensional settings.

Future Work

- **High-Dimensional Extensions:** Apply the solver to high-dimensional PDEs where classical methods fail.
- **Nonlinear PDEs:** Extend the method using backward SDEs and stochastic control techniques.
- **Variance Reduction:** Incorporate techniques like importance sampling to improve training efficiency.
- **Financial Applications:** Test the method on exotic and path-dependent derivatives for real-world relevance.

Thank You

Questions?