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

Shubha Sanket Samantaray
Université Grenoble Alpes

May 16, 2025

**Abstract**

Partial Differential Equations (PDEs) play a fundamental role in physics, engineering, and mathematical finance but solving high-dimensional PDEs remains a significant computational challenge. Traditional numerical methods, such as finite difference and finite element methods, suffer from the curse of dimensionality. In this paper, we explore a neural network-based approach to solving PDEs, leveraging the probabilistic representation given by the Feynman-Kac theorem. By reformulating the PDE problem into an equivalent stochastic representation, we use Neural Networks to approximate solutions efficiently. We discuss the theoretical foundations of this approach, its numerical implementation, benchmark its performance against classical solvers and offer a scalable alternative to conventional solvers.

## 1 Introduction

Partial Differential Equations (PDEs) play a central role in modeling a wide range of phenomena across the physical sciences, biology, finance, and engineering. Whether describing heat flow, chemical diffusion, population dynamics, or stochastic control systems, PDEs provide a mathematical framework for understanding how quantities evolve under deterministic or random influences.

Despite their broad applicability, solving PDEs, especially in high-dimensional settings, remains a significant computational challenge. Classical numerical methods such as finite differences, finite elements, and spectral methods are well-established for low-dimensional problems, but they often become infeasible as dimensionality increases, a phenomenon known as the curse of dimensionality.

An alternative approach is offered by probabilistic methods, which reformulate certain classes of PDEs in terms of expectations over stochastic processes. These representations, often grounded in the Feynman–Kac framework, allow solutions to be expressed as conditional expectations. This shift from solving deterministic equations to estimating statistical quantities opens the door to modern machine learning techniques.

In recent years, neural networks have emerged as powerful tools for high-dimensional function approximation. Their flexibility and expressiveness make them natural candidates for approximating the conditional expectations that arise in probabilistic PDE representations. Furthermore, their training objectives align closely with statistical estimation principles, such as minimizing mean squared error, which coincides with the optimal prediction under squared-loss risk.

This paper explores the use of neural networks to approximate solutions to linear parabolic PDEs via their probabilistic representations. By combining stochastic simulation methods with neural network regression, we aim to develop a mesh-free, scalable alternative to classical PDE solvers. We begin with the heat equation as a foundational test case, and then extend the methodology to more complex equations, illustrating the strengths and limitations of the approach through numerical experiments.

Ultimately, this work demonstrates how combining tools from stochastic analysis and deep learning can offer effective strategies for solving PDEs in settings where traditional methods fall short. Our results suggest that neural network-based methods provide a promising direction for high-dimensional problems that naturally admit a probabilistic reformulation.

# 2 Theoretical Background

## 2.1 Heat Equation in the Forward and Backward Forms

We consider the following heat equation in the backward form, where for a function $u : [0, T] \times \mathbb{R}^d \to \mathbb{R}$, we have

$$\partial_t u + \frac{1}{2}\Delta u = 0 \quad \text{on } [0, T) \times \mathbb{R}^d, \quad u \in \mathcal{C}^{1,2}, \tag{1}$$

equipped with the terminal condition

$$u(T, \cdot) = f(\cdot), \tag{2}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a given function.
This problem is companion to the heat equation in the forward form through the following change of variables:

$$v(t, x) := u(T - t, x), \tag{3}$$

for all $(t, x) \in [0, T] \times \mathbb{R}^d$ where $v : [0, T] \times \mathbb{R}^d \to \mathbb{R}$. This transformation reverses the time parameter and defines a new function $v$ on the interval $[0, T]$.
To determine the partial differential equation satisfied by $v$, we compute its derivatives. Differentiating $v$ with respect to $t$, we have

$$\partial_t v(t, x) = -\partial_t u(T - t, x). \tag{4}$$

Since the transformation does not affect the spatial variable, the Laplacian satisfies

$$\Delta v(t, x) = \Delta u(T - t, x). \tag{5}$$

Substituting these into the equation satisfied by $u$, we obtain

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

which, expressed in terms of $v$, becomes

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

being equivalent to

$$\partial_t v(t, x) - \frac{1}{2}\Delta v(t, x) = 0. \tag{6}$$

Next, we compute the initial condition for $v$. By the definition of $v$, we have

$$v(0, x) = u(T, x) = f(x) \tag{7}$$

We have thus shown that the function $v$, defined by the transformation $v(t, x) = u(T - t, x)$, satisfies the partial differential equation

$$\partial_t v - \frac{1}{2}\Delta v = 0 \quad \text{on } (0, T] \times \mathbb{R}^d, \quad v \in \mathcal{C}^{1,2}, \tag{8}$$

with the initial condition

$$v(0, \cdot) = f(\cdot) \quad \text{on } \mathbb{R}^d. \tag{9}$$

It is equally possible to begin with the function $v$, defined as the solution to equations (8) and (9), and recover a function $u$ satisfying (1) and (2) by proceeding with the same time-reversal arguments. This bidirectional correspondence, coupled with the uniqueness of solutions to the respective PDEs, establishes that the two formulations are indeed equivalent.

## 2.2 Feynman–Kac Formula of the Heat Equation in the Backward Form

Our goal in this section is to derive an explicit probabilistic representation for the solution $u(t, x)$ of the partial differential equation

$$\partial_t u + \frac{1}{2}\Delta u = 0 \quad \text{on } [0, T) \times \mathbb{R}^d, \qquad u(T, \cdot) = f(\cdot),$$

by expressing it as the conditional expectation of a functional of Brownian motion. This representation will ultimately take the form of the Feynman–Kac formula. To fix ideas, we treat the case $d = 1$. Note that $\Delta u = \partial_{xx} u = u_{xx}$, and we use these notations interchangeably throughout the derivation.

The Itô–Döblin formula for Brownian motion (see Theorem A.2.6 in Appendix) in its differential form states that for a function $f(t, W_t)$:

$$df(t, W_t) = f_t(t, W_t)dt + f_x(t, W_t)dW_t + \frac{1}{2} f_{xx}(t, W_t)dt. \tag{10}$$

Applying this to our problem, we rewrite:

$$du(t, W_t) = \left[ u_t(t, W_t) + \frac{1}{2} u_{xx}(t, W_t) \right] dt + u_x(t, W_t)dW_t. \tag{11}$$

Since in our PDE, the term inside the brackets is equal to zero, the expression above simplifies to:

$$du(t, W_t) = u_x(t, W_t)dW_t. \tag{12}$$

Let $t \leq s \leq T$. We integrate both sides from $t$ to $s$:

$$u(s, W_s) = u(t, W_t) + \int_t^s \partial_x u(r, W_r)dW_r. \tag{13}$$

The integral at hand represents an Itô integral. As the Itô integral is a martingale, we know that the expectation of the Itô integral starting from 0 at time $t$ vanishes, therefore:

$$\mathbb{E}\left[ \int_t^s \partial_x u(r, W_r)dW_r \mid W_t = x \right] = \mathbb{E}\left[ \int_t^t \partial_x u(r, W_r)dW_r \mid W_t = x \right] = \mathbb{E}\left[ 0 \mid W_t = x \right] = 0, \tag{14}$$

as we have

$$\int_t^t \partial_x u\big(r, W_r^x\big)\, dW_r = 0 \quad \text{knowing } W_t = x.$$

Another way to justify this point is to use **Itô isometry** (see Theorem A.2.3 in Appendix), which provides the following identity:

$$\mathbb{E}\left[ I^2(s) \right] = \mathbb{E}\left[ \int_t^s \left( \Delta u(W_r) \right)^2 dr \right], \quad \text{where } I(s) := \int_t^s \Delta u(W_r)\, dW_r. \tag{15}$$

Taking expectation at time $s = t$, we get:

$$\mathbb{E}\left[ I^2(t) \right] = \mathbb{E}\left[ \int_t^t \Delta^2 u\, dr \right] = 0. \tag{16}$$

Since $I^2(t) \geq 0$, it follows that $\mathbb{E}\left[ I^2(t) \right] = 0 \implies I^2(t) = 0 \implies I(t) = 0$.
Now taking conditional expectations on both sides of equation (13), we obtain:

$$u(t, x) = \mathbb{E}\left[ u(s, W_s) \mid W_t = x \right]. \tag{17}$$

for $t \in [0, T)$ and any $s \in [t, T)$.

In particular, taking the limit as $s \to T$, we exploit the regularity assumption $u \in \mathcal{C}^{1,2}$, which ensures continuity of the map $(t, x) \mapsto u(t, x)$. Since $W_s \to W_T$ almost surely and $u(s, W_s) \to u(T, W_T)$ pointwise, and assuming $u$ is bounded or satisfies suitable growth conditions, we may apply the Dominated Convergence Theorem to pass the limit inside the conditional expectation. We therefore obtain:

$$u(t, x) = \lim_{s \to T} \mathbb{E}\left[ u(s, W_s) \mid W_t = x \right] = \mathbb{E}\left[ u(T, W_T) \mid W_t = x \right]. \tag{18}$$

Since we imposed $u(T, W_T) = f(W_T)$, the final result follows:

$$u(t, x) = \mathbb{E}\left[ f(W_T) \mid W_t = x \right]. \tag{19}$$

We have thus derived the **Feynman-Kac** representation for the solution $u$ of the PDE.

## 2.3 Feynman-Kac Formula of the Heat Equation in the Forward Form

We now aim to solve the **forward heat equation** given by:

$$\partial_t v - \frac{1}{2}\Delta v = 0 \quad \text{on } (0,T] \times \mathbb{R}. \tag{20}$$

with the initial condition:

$$v(0,\cdot) = f(\cdot) \quad \text{on } \mathbb{R}. \tag{21}$$

We seek to show that the solution satisfies the **probabilistic** representation:

$$v(t,x) = \mathbb{E}^x \left[ f(W_t) \right], \tag{22}$$

where $\mathbb{E}^x(\cdot)$ is the expectation conditioned on $W_0 = x$:

$$\mathbb{E}^x(\cdot) = \mathbb{E}\left[ \cdot \mid W_0 = x \right]. \tag{23}$$

To prove this result, we recall the **homogeneous Markov property** of Brownian motion.

Let $(X_t)_{t\geq 0}$ be a stochastic process adapted to a filtration $(\mathcal{F}_t^X)_{t\geq 0}$, where $\mathcal{F}_t^X := \sigma(X_s : 0 \leq s \leq t)$ represents the natural filtration generated by the process $X$ up to time $t$. The Markov property asserts that, for any $0 \leq s < t$,

$$\mathbb{E}\left[ f(X_t) \mid \mathcal{F}_s^X \right] = \mathbb{E}\left[ f(X_t) \mid X_s \right], \tag{24}$$

for any bounded measurable function $f$.

In other words, the conditional distribution of the future state $X_t$, given the past up to time $s$, depends only on the present state $X_s$, and not on the full history.

In addition, if $X$ is a *homogeneous Markov process*, then for all $0 \leq s < t$ and $x \in \mathbb{R}^d$, we have:

$$\mathbb{E}\left[ f(X_t) \mid X_s = x \right] = \mathbb{E}\left[ f(X_{t-s}) \mid X_0 = x \right], \tag{25}$$

where the right-hand side expresses the fact that the transition probabilities depend only on the elapsed time $t - s$, and not on the absolute time points. This captures the defining feature of *time-homogeneity*.

We now apply the homogeneous Markov property to the Brownian motion $W_t$ in order to derive a probabilistic representation for the function $v(t,x)$.

From the time-reversal transformation we recall that:

$$v(t,x) = u(T-t,x) = \mathbb{E}\left[ f(W_T) \mid W_{T-t} = x \right]. \tag{26}$$

Invoking the time-homogeneous Markov property of Brownian motion we have:

$$\mathbb{E}\left[ f(W_T) \mid W_{T-t} = x \right] = \mathbb{E}\left[ f(W_{T-(T-t)} \mid W_0 = x \right] = \mathbb{E}\left[ f(W_t) \mid W_0 = x \right].$$

Hence, we arrive at the desired representation of the solution:

$$v(t,x) = \mathbb{E}\left[ f(W_t) \mid W_0 = x \right]. \tag{27}$$

Thus, the forward heat equation satisfies the probabilistic representation:

$$v(t,x) = \mathbb{E}^x \left[ f(W_t) \right]. \tag{28}$$

## 2.4 Feynman–Kac Representation for General Linear Parabolic PDEs

We now aim to prove that the solution $u(t,x)$ of the linear parabolic PDE

$$\partial_t u(t,x) + \frac{1}{2}\sigma^2(x)\partial_{xx} u(t,x) + b(x)\partial_x u(t,x) - r u(t,x) = 0, \quad \forall (t,x) \in [0,T) \times \mathbb{R}, \tag{29}$$

with terminal condition

$$u(T,x) = f(x), \quad \forall x \in \mathbb{R}, \tag{30}$$

admits the Feynman–Kac representation

$$u(t,x) = \mathbb{E}\left[e^{-r(T-t)} f(X_T) \mid X_t = x\right], \quad \forall (t,x) \in [0,T] \times \mathbb{R}. \tag{31}$$

where $X(s)$ satisfies the SDE:

$$dX(s) = b(X(s))\, ds + \sigma(X(s))\, dW(s) \tag{32}$$

and $\mathbb{E}[\cdot \mid X_t = x]$ denotes the expectation conditioned on $X_t = x$.

Note that throughout the problem setup and the derivation that follows, the notations $X(t)$ and $X_t$ are used interchangeably and refer to the same quantity.

Let us fix $t_0 \in [t, T)$, define

$$g(t, X(t)) := e^{-r(t-t_0)} u(t, X(t)),$$

and determine how the multidimensional Itô formula is used to obtain the desired result.

We define a two-dimensional process

$$X_t := (X_t^0, X_t^1) := (t, X(t)).$$

1. $X_t^0 = t$ is a continuous finite variation process, hence a valid semimartingale.
2. $X_t^1 = X(t) = \xi + \int_{t_0}^t b(X(s))\, ds + \int_{t_0}^t \sigma(X(s))\, dW(s)$ is also a semimartingale, as shown below:
   (a) The process $\int_{t_0}^t b(X(s))\, ds$ is of finite variation because for almost every path of $X_s$, the function $t \mapsto \int_{t_0}^t b(X_s)\, ds$ is of class $\mathcal{C}^1$.
   (b) From Itô isometry and under suitable assumptions on $\sigma^2$ (see [22]),

$$\mathbb{E}\left[\left(\int_{t_0}^t \sigma(X(s))\, dW(s)\right)^2\right] = \mathbb{E}\left[\int_{t_0}^t \sigma^2(X(s))\, ds\right] < +\infty,$$

   so $\int_{t_0}^t \sigma(X(s))\, dW(s)$ is a square-integrable martingale, and thus a local martingale.

Hence, $\int_{t_0}^t b(X(s))\, ds + \int_{t_0}^t \sigma(X(s))\, dW(s)$ is a semimartingale.

Furthermore, for any semimartingale $Y(t)$ and a random variable $\xi$, the process defined by $Y(t) + \xi$ is a semimartingale, therefore the process $X(t) = \xi + \int_{t_0}^t b(X(s))\, ds + \int_{t_0}^t \sigma(X(s))\, dW(s)$ is a semimartingale.

The $C^2$ nature of the function $u$ is evident and has therefore been omitted from the discussion. The conditions of the Itô formula are thus satisfied.

Let us now derive the Feynman–Kac representation of the PDE using the multidimensional Itô formula (see Theorem A.3.2 in Appendix). Applying it to $g(t, X(t)) = e^{-r(t-t_0)} u(t, X(t))$, we have:

$$\begin{aligned}
dg(t, X(t)) &= d\left(e^{-r(t-t_0)} u(t, X(t))\right) \\
&= \left[-re^{-r(t-t_0)} u(t, X(t)) + e^{-r(t-t_0)} \partial_t u(t, X(t))\right] dt \\
&\quad + e^{-r(t-t_0)} \partial_x u(t, X(t))\, dX(t) \\
&\quad + \frac{1}{2}\left(2\partial_{tx} g(t, X(t))\, d\langle \mathrm{id}, X\rangle_t + \partial_{tt} g(t, X(t))\, d\langle \mathrm{id}, \mathrm{id}\rangle_t + \partial_{xx} g(t, X(t))\, d\langle X, X\rangle_t\right).
\end{aligned}$$

Now, since $d\langle \mathrm{id}, X\rangle_t = d\langle t, X(t)\rangle = 0$, $d\langle \mathrm{id}, \mathrm{id}\rangle_t = d\langle t, t\rangle = 0$, and from Lemma A.2.8,

$$d\langle X, X\rangle_t = \sigma^2(X(t))\, dt,$$

the formula simplifies to:

$$\begin{aligned}
dg(t, X(t)) &= \left[-re^{-r(t-t_0)} u(t, X(t)) + e^{-r(t-t_0)} \partial_t u(t, X(t))\right] dt \\
&\quad + e^{-r(t-t_0)} \partial_x u(t, X(t))\, dX(t) \\
&\quad + \frac{1}{2} e^{-r(t-t_0)} \partial_{xx} u(t, X(t))\, \sigma^2(X(t))\, dt.
\end{aligned}$$

Substituting the SDE

$$dX(t) = b(X(t))\,dt + \sigma(X(t))\,dW(t),$$

into the above expression yields:

$$\begin{aligned} dg(t,X(t)) &= e^{-r(t-t_0)}\partial_x u(t,X(t))\sigma(X(t))\,dW(t) \\ &\quad + e^{-r(t-t_0)}\left[\partial_t u(t,X(t)) + b(X(t))\partial_x u(t,X(t)) + \frac{1}{2}\sigma^2(X(t))\partial_{xx}u(t,X(t)) - ru(t,X(t))\right]dt. \end{aligned}$$

Now, from (29) we have:

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

so we obtain:

$$dg(t,X(t)) = e^{-r(t-t_0)}\partial_x u(t,X(t))\sigma(X(t))\,dW(t).$$

Integrating this expression between $t_0$ and $t \in [t_0, T)$, we get:

$$e^{-r(t-t_0)}u(t,X(t)) = u(t_0,X(t_0)) + \int_{t_0}^{t} e^{-r(s-t_0)}\partial_x u(s,X(s))\sigma(X(s))\,dW(s).$$

Taking the conditional expectation with respect to $X_{t_0} = x$, we obtain:

$$\begin{aligned} \mathbb{E}\left[e^{-r(t-t_0)}u(t,X(t)) \mid X_{t_0} = x\right] &= \mathbb{E}\left[u(t_0,X(t_0)) + \int_{t_0}^{t} e^{-r(s-t_0)}\partial_x u(s,X(s))\sigma(X(s))\,dW(s) \mid X_{t_0} = x\right] \\ &= u(t_0,x) + \mathbb{E}\left[\int_{t_0}^{t} e^{-r(s-t_0)}\partial_x u(s,X(s))\sigma(X(s))\,dW(s) \mid X_{t_0} = x\right]. \end{aligned}$$

Since the Itô integral is a martingale, we have:

$$\begin{aligned} \mathbb{E}\left[\int_{t_0}^{t} \partial_x u(s,X(s))\sigma(X(s))\,dW(s) \mid X_{t_0} = x\right] &= \mathbb{E}\left[\int_{t_0}^{t_0} \partial_x u(s,X(s))\sigma(X(s))\,dW(s) \mid X_{t_0} = x\right] \\ &= 0. \end{aligned}$$

Thus,

$$\mathbb{E}\left[e^{-r(t-t_0)}u(t,X(t)) \mid X_{t_0} = x\right] = u(t_0,x).$$

Taking $t \to T$ and using the same convergence arguments as those presented in section 2.3, we get:

$$\mathbb{E}\left[e^{-r(T-t_0)}u(T,X(T)) \mid X_{t_0} = x\right] = u(t_0,x),$$

and hence:

$$u(t,x) = \mathbb{E}\left[e^{-r(T-t)}f(X_T) \mid X_t = x\right] \quad \text{for any } (t,x) \in [0,T) \times \mathbb{R}$$

along with the obvious case of $u(T,x) = f(x) = \mathbb{E}\left[f(X_T) \mid X_T = x\right] \quad \forall x \in \mathbb{R}$.

## 2.5 The Black–Scholes PDE and its Probabilistic Representation

The Black–Scholes model provides a fundamental equation for the pricing of European options. An *option* is a financial contract that gives its holder the right, but not the obligation, to buy (call) or sell (put) an asset at a fixed price $K$ (called the *strike price*) at a predetermined maturity time $T$. The *payoff* of an option is the amount received at maturity, depending on the price of the underlying asset. Its *price* at a given time $t < T$ is determined by arbitrage arguments and satisfies the Black–Scholes PDE (see [13]).

Let $V(t,S)$ denote the price of an option at time $t \in [0,T)$, when the underlying asset has price $S > 0$. The Black–Scholes PDE reads:

$$\partial_t V(t,S) + rS\,\partial_S V(t,S) + \frac{1}{2}\sigma^2 S^2\,\partial_{SS}V(t,S) - rV(t,S) = 0, \tag{33}$$

with terminal condition:

$$V(T,S) = \Phi(S), \tag{34}$$

where:

- $r > 0$: constant risk-free interest rate;
- $\sigma > 0$: volatility of the asset;
- $\Phi(S)$: payoff function of the option at maturity $T$;

When the terminal condition corresponds to a European call option with strike $K > 0$, i.e.,

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

the solution to (33)–(34) is given by the Black–Scholes formula:

$$V(t, S) = S\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2), \tag{35}$$

where $\Phi(\cdot)$ denotes the standard normal cumulative distribution function, and

$$d_1 = \frac{\log(S/K) + (r + \sigma^2/2)(T - t)}{\sigma\sqrt{T - t}},$$
$$d_2 = d_1 - \sigma\sqrt{T - t}.$$

This closed-form expression provides an efficient and widely used method for computing the fair price of European options under the Black–Scholes model.

**Connection to General Linear Parabolic PDEs.** The Black–Scholes equation is a special case of the general second-order linear parabolic PDE:

$$\partial_t u + b(x)\,\partial_x u + \frac{1}{2}\sigma^2(x)\,\partial_{xx} u - ru = 0, \tag{36}$$

with the identifications:

$$x = S, \quad b(S) = rS, \quad \sigma(S) = \sigma S, \quad \text{so that } \sigma^2(S) = \sigma^2 S^2.$$

Therefore, the Black–Scholes PDE fits directly into the Feynman–Kac framework.

**Probabilistic Representation Using Feynman–Kac.** Let $(S_t)_{t\in[0,T]}$ evolve under the risk-neutral measure $\mathbb{Q}$ as:

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t^{\mathbb{Q}}. \tag{37}$$

Then, by the Feynman–Kac formula, the solution to (33)–(34) admits the representation:

$$V(t, S) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}\Phi(S_T) \mid S_t = S\right]. \tag{38}$$

In particular, for a European call option,

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

Hence, the Black–Scholes PDE is not only a canonical model in financial mathematics but also a concrete example of the general class of linear parabolic PDEs whose solutions admit Feynman–Kac representations.

## 2.6 Neural Networks as Conditional Expectation Approximators

In light of the Feynman–Kac representation of solutions to linear parabolic PDEs, our objective is to approximate conditional expectations of the form

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

using function approximation methods. A natural framework for this is **regression**, where the goal is to learn a function that best predicts the output $f(X_T)$ based on input $X_t$.

Formally, we consider the following optimization problem:

$$\varphi^\star = \arg\min_{\varphi} \mathbb{E}\left[\left(f(X_T) - \varphi(X_t)\right)^2\right], \tag{39}$$

where the minimization is over all measurable functions $\varphi$ of $X_t$. This corresponds to projecting $f(X_T)$ onto the space of measurable functions of $X_t$ in $L^2$, and the unique solution is known to be the conditional expectation:

$$\varphi^\star(x) = \mathbb{E}\left[f(X_T) \mid X_t = x\right]. \tag{40}$$

This theoretical characterization aligns exactly with the probabilistic solutions derived via the Feynman–Kac formula. The conditional expectation minimizing the mean squared error coincides with the value of the function solving the PDE.

To approximate $\varphi^\star$, we employ a **neural network** $\varphi_\theta$, parameterized by weights $\theta$, and train it to minimize the empirical version of the loss:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f(X_T^i) - \varphi_\theta(X_t^i)\right)^2. \tag{41}$$

Here, $(X_t^i, X_T^i)$ are samples from the Itô process. By the **universal approximation theorem**, neural networks can approximate any square-integrable function arbitrarily well, assuming sufficient depth and width.

In essence, the regression framework implicitly mirrors the Feynman–Kac structure: both involve computing a conditional expectation of a terminal payoff given the current state. Our strategy leverages this equivalence. By training a neural network to minimize the squared loss on simulated data from the underlying stochastic process, we effectively learn the solution to the original PDE. This connection allows us to recast the PDE-solving problem as a supervised learning task: the neural network learns to approximate the mapping $x \mapsto \mathbb{E}[f(X_T) \mid X_t = x]$, thus providing a powerful and scalable method for solving high-dimensional parabolic PDEs.

# 3 Methodology and Implementation

This section presents the numerical and algorithmic framework developed to approximate solutions to linear parabolic PDEs using neural networks through their Feynman–Kac representations. The implementation is carried out in Python, with full source code provided in the Jupyter notebook `Feynman_Kac_Solver.ipynb`.
We progressively examine three cases of increasing complexity to validate the method and explore its generalization.

## 3.1 Forward Heat Equation as a Benchmark Problem

We begin with the classical forward heat equation:

$$\partial_t u(t,x) = \frac{1}{2}\partial_{xx} u(t,x), \qquad u(0,x) = x^2,$$

whose exact solution is given by $u(t,x) = x^2 + t$. The Feynman–Kac formula tells us that this solution can be written as:

$$u(t,x) = \mathbb{E}[f(W_t) \mid W_0 = x], \quad \text{with } f(x) = x^2,$$

where $(W_t)_{t\geq 0}$ is Brownian motion starting from $x$.

**Data Generation:** For a fixed time $t$, we simulate $N$ samples of $(W_0^i, W_t^i)$ as follows:

- Sample $W_0^i \sim \mathcal{U}[a,b]$,
- Generate $W_t^i = W_0^i + \sqrt{t}\,\xi^i$, with $\xi^i \sim \mathcal{N}(0,1)$,
- Set the target as $y^i = f(W_t^i) = (W_t^i)^2$.

This yields the dataset $\mathcal{D} = \{(W_0^i,\, (W_t^i)^2)\}_{i=1}^N$.

**Learning Objective:** A one-dimensional neural network $\varphi_\theta$ is trained to approximate the conditional expectation $\mathbb{E}[f(W_t) \mid W_0 = x]$ by minimizing the empirical mean squared error (MSE):

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left( (W_t^i)^2 - \varphi_\theta(W_0^i) \right)^2 .$$

## 3.2 Pricing European Call Options via the Black–Scholes Equation

We next turn to the problem of pricing a European call option using the Black–Scholes PDE. The governing equation is:

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

where the terminal condition is the payoff of the option, $\Phi(S) = (S - K)^+$. Under the risk-neutral measure $\mathbb{Q}$, the solution admits the probabilistic representation:

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

where the underlying asset price $S_t$ follows the geometric Brownian motion:

$$dS_t = rS_t \, dt + \sigma S_t \, dW_t. \tag{42}$$

**Simulation Schemes for $S_T$:** To train the neural network to approximate the option price $V(t, S)$, we generate target labels $y^i = \Phi(S_T^i)$ by simulating the terminal stock price $S_T^i$ starting from $S_t^i = x^i$, using two different schemes:

- **Closed-form simulation:** The SDE (42) admits an explicit solution, namely the geometric Brownian Motion:
$$S_T = x \cdot \exp\left( \left( r - \frac{1}{2}\sigma^2 \right) T + \sigma \cdot W_T \right), \quad \text{where } x = S_0.$$
and we use the following equation to generate our training samples:
$$S_T = x \cdot \exp\left( \left( r - \frac{1}{2}\sigma^2 \right) (T - t) + \sigma\sqrt{T - t} \cdot Z \right), \quad x = S_t \text{ and } Z \sim \mathcal{N}(0, 1).$$

- **Euler–Maruyama discretization:** To assess the behavior of the learning method under numerical SDE simulation, we also implement the Euler–Maruyama scheme. The time interval $[t, T]$ is divided into $M$ equal steps of size $\Delta t$, and the asset price is evolved iteratively.

  Starting from $S_t = x$, we update the value at each step using:

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

  for $n = 0, 1, \ldots, M - 1$. This recursive scheme accumulates both drift and diffusion increments over time, yielding an approximation of $S_T$ at the final step.

**Training Dataset:** To build the training dataset for the neural regression model:

- We uniformly sample initial asset prices $x^i = S_t^i \in [a, b] \subset \mathbb{R}_+$,
- We simulate the terminal stock price $S_T^i$ using either of the two methods above,
- We compute the discounted payoff $y^i = e^{-r(T-t)}(S_T^i - K)^+$,
- We form the dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$.

This dual-scheme approach allows us to compare the accuracy of solutions under both exact and approximate sampling of the underlying SDE, providing insight into the robustness of the method. It also provides us with key insights into the extension of this solver to PDEs which do not have a closed-form solution.

### 3.3 Importance of Fixed-Time Conditioning

A critical requirement in the Feynman–Kac framework is that the conditional expectation is taken with respect to $X_t = x$ at a fixed time $t$. To correctly learn the mapping $x \mapsto u(t, x)$, we must:

- Keep time $t$ fixed during data generation,
- Vary $x$ spatially and simulate $X_T^i \mid X_t^i = x^i$,

This conditioning must be strictly respected for the neural network solver to produce accurate results, as in the one-dimensional heat equation setup and in the pricing of European call options under the Black–Scholes model, where the regression target corresponds to a conditional expectation at a fixed time.

### 3.4 Approximating the Full Space-Time Solution Surface

Having validated the method for fixed $t$ and variable $x$, we now extend the approach to learn the full solution map:

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

by training a two-input neural network $\varphi_\theta(t, x)$ that takes both time and spatial coordinates as input.

In the fixed-time setting, the neural network approximates the mapping $x \mapsto V(t, x)$, i.e., a slice of the solution surface at a given time. While this aligns exactly with the structure of the Feynman–Kac formula, it requires retraining for each new time value $t$. To overcome this limitation, we extend the input space to include both time and space, enabling the network to learn the entire solution surface simultaneously.

The key idea is to investigate whether a neural network can learn a family of conditional expectations across a continuous range of time values. This enhances the model's generality and reusability, as one network can then evaluate $V(t, x)$ for arbitrary $(t, x) \in [0, T] \times \mathbb{R}_+$ without additional retraining.

**Sampling and Training Procedure.** To train this 2D neural network, we generate a dataset as follows:

- Sample pairs $(t^i, x^i) \in [0, T] \times [a, b]$ uniformly or from a structured grid,
- Given $S_{t^i} = x^i$, simulate $S_T^i$ using the following equation:

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

  or by using Euler-Maruyama discretization.

- Compute the target value:

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

- Store the training pair $((t^i, x^i), y^i)$.

The neural network is then trained to minimize the empirical MSE loss:

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

**Experimental Nature of the Extension.** Unlike the fixed-time experiments, where the conditional expectation has a well-defined probabilistic interpretation for each $t$, this setup implicitly requires the neural network to learn a continuous family of such expectations. From a theoretical perspective, this extension is not directly guaranteed by the Feynman–Kac formula, which conditions on a fixed time.

### 3.5 Neural Network Architecture and Optimization

The supervised learning problems encountered in this paper differ fundamentally from conventional neural network training tasks in function approximation. In a standard setting—say, approximating the function $f(x) = \sin(x)$—the training data consists of deterministic input–output pairs $(x_i, \sin(x_i))$. Each input $x_i$ deterministically maps to a unique label $y_i = f(x_i)$.

In contrast, our setting involves learning a function $\varphi(t, x) \approx \mathbb{E}[f(X_T) \mid X_t = x]$, where $X_t$ is the solution of a SDE. The crucial difference is that the target $y = f(X_T)$ is a *random variable* even for fixed $(t, x)$. Thus, multiple simulations starting from the same point $(t, x)$ can yield different terminal values $X_T$, and hence different observed outputs $y$. The conditional expectation we seek is the mean of this distribution—not a deterministic value associated to each input.

Therefore, the training set

$$\mathcal{D} = \left\{ \left( (t^i, x^i),\, y^i = f(X_T^i) \right) \right\}_{i=1}^{N}$$

contains stochastic samples from the conditional distribution of $f(X_T) \mid X_t = x$, where $X_T^i$ is a Monte Carlo realization starting from $x^i$ at time $t^i$. The neural network does not observe the true conditional expectation directly; it must infer it from noisy samples. This distinguishes our regression target from the deterministic labels used in conventional supervised learning.

**Architecture.** For all experiments, we use a fully-connected feedforward neural network with:

- **Input dimension:** 1 (for fixed-time regression $x \mapsto \varphi(t, x)$) or 2 (for full-surface regression $(t, x) \mapsto \varphi(t, x)$),
- **Hidden layers:** Three layers with 64, 32, and 32 neurons respectively, each followed by a Tanh activation,
- **Output dimension:** 1, corresponding to the estimated value $\varphi_\theta(t, x) \approx \mathbb{E}[f(X_T) \mid X_t = x]$.

**Loss Function and Optimization.** We train the network using the empirical mean squared error (MSE) loss:

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

where the labels $y^i$ are generated by simulating the terminal value $X_T^i$ and applying the payoff function $f$. As explained above, the randomness of $y^i$ reflects the stochasticity inherent in the underlying SDE.

We use the Adam optimizer with a learning rate $\eta = 10^{-5}$, and train using mini-batch stochastic gradient descent over several epochs. Given the noise inherent in the training labels, the batch size and number of samples $N$ play an important role in reducing the variance of the gradient estimates and stabilizing training.

In summary, this framework demonstrates that the Feynman–Kac representation transforms the PDE problem into a statistical regression problem, solvable using neural networks trained on simulated stochastic paths. This approach scales flexibly and avoids mesh-based discretization, making it suitable for high-dimensional PDE problems.

## 4 Results and Discussion

This section presents and analyzes the results obtained from training neural networks to approximate PDE solutions via their Feynman–Kac representations. We evaluate each experimental setting in terms of approximation accuracy, agreement with known analytical solutions, and overall qualitative behavior. Quantitative comparisons and visualizations are included to support the analysis.

### 4.1 Heat Equation: Validation of the Feynman–Kac Solver

We train our model by setting the time to $t = 1$ in (20).

**Simulated Data.** Figure 1 shows the generated data used for training. On the left, we plot the Brownian samples $(W_0, W_t)$, and on the right, we show the histogram of the target values $u(0, W_t)$. The distribution is positively skewed as expected from squaring Gaussian samples.

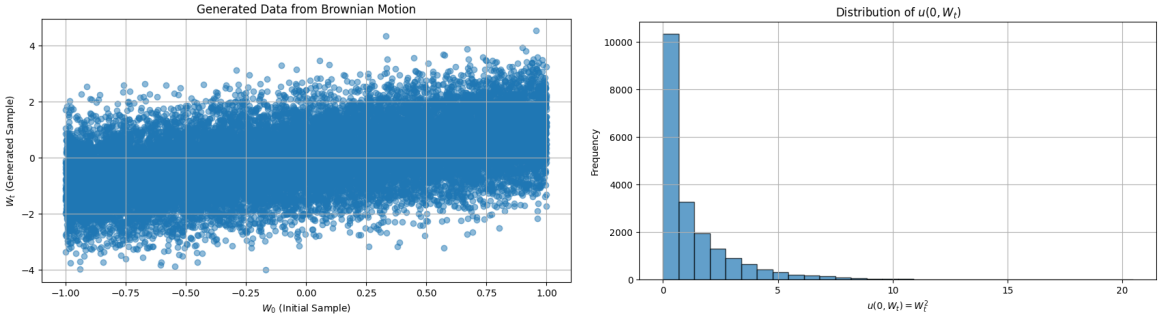


Figure 1: Left: Scatter plot of samples $(W_0, W_t)$. Right: Histogram of $u(0, W_t) = W_t^2$.

**Training Performance.** The network was trained using the Adam optimizer for 50,000 epochs. The left plot in Figure 2 shows the training loss, which stabilizes after initial transients. The right plot shows the predicted values $\varphi_\theta(W_0)$ against the true noisy targets $u(0, W_t)$, indicating that the network captures the correct trend despite label noise.

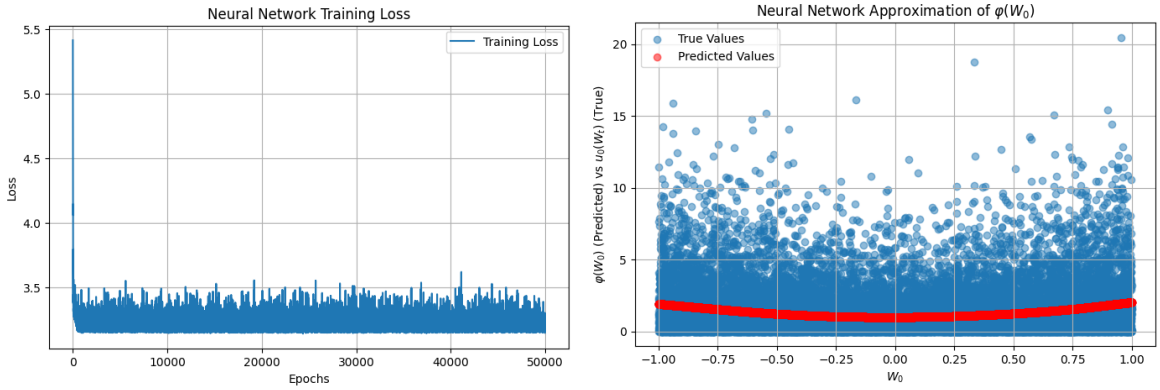


Figure 2: Left: Neural network training loss. Right: Predicted $\varphi(W_0)$ vs noisy targets $u(0, W_t)$.

**Comparison with True Function.** To verify the network output, we compare $\varphi_\theta(W_0)$ against the analytical solution $u(1, W_0) = W_0^2 + 1$. As shown in Figure 3, the approximation aligns very closely with the true function. The right panel shows the absolute pointwise error, which remains below 0.05 across most of the domain.

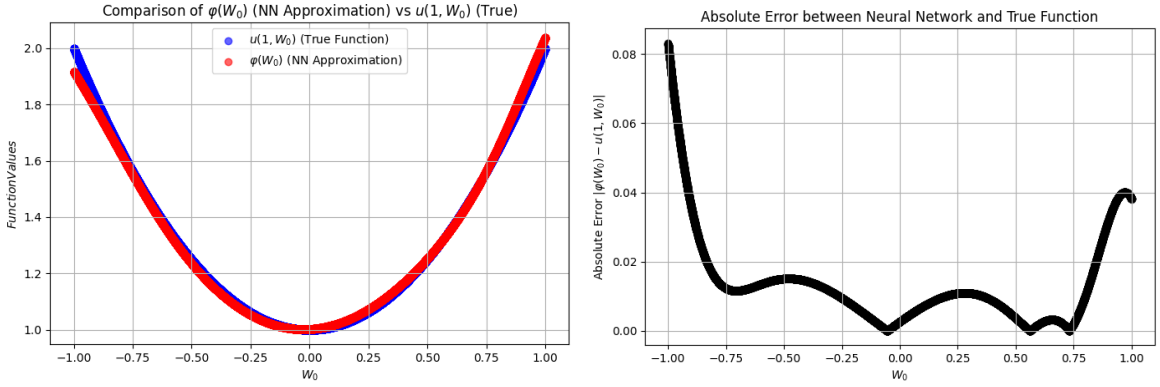


Figure 3: Left: Comparison of $\varphi(W_0)$ with exact $u(1, W_0)$. Right: Absolute error $|\varphi(W_0) - u(1, W_0)|$.

The results confirm that our neural network solver successfully recovers the analytical solution of the heat equation. The network learns the correct conditional expectation, with training loss stabilizing over epochs, predictions matching the true function $u(1, x) = x^2 + 1$, and absolute errors remaining uniformly small across the input domain.

## 4.2 European Call Option: Fixed-Time Evaluation

We test our solver on the Black–Scholes PDE for a European call option at a fixed time $t = 0.5$. The asset follows a geometric Brownian motion, and the payoff is $\Phi(S_T) = (S_T - K)^+$. Two data generation schemes are considered: one using the Euler–Maruyama discretization of the SDE, and the other based on the closed-form solution of $S_T$.

**Euler–Maruyama Results.** The network is trained on the data generated via Euler–Maruyama for 5000 epochs. Figure 4 shows the training loss convergence and the comparison between the predicted option price and the analytical Black–Scholes formula at $t = 0.5$. The neural network learns the correct functional form of the price surface with a mean absolute error of 0.001482 and maximum error of 0.005502.

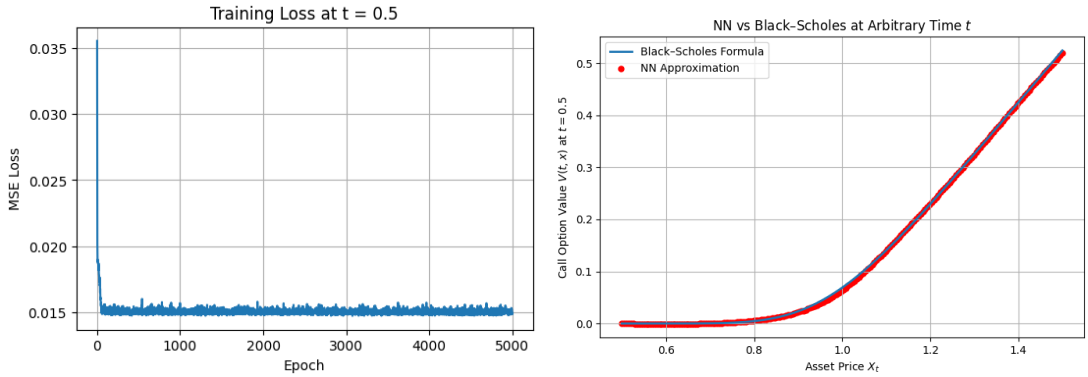


Figure 4: Euler–Maruyama results. Left: MSE training loss over 5000 epochs. Right: Neural network prediction vs. Black–Scholes formula at $t = 0.5$.

**Closed-Form Sampling Results.** We repeat the experiment using the exact solution of geometric Brownian motion to generate $S_T$. As shown in Figure 5, the network again accurately learns the Black–Scholes pricing map, achieving even lower approximation errors: a mean absolute error of 0.001085 and a maximum error of 0.003068.

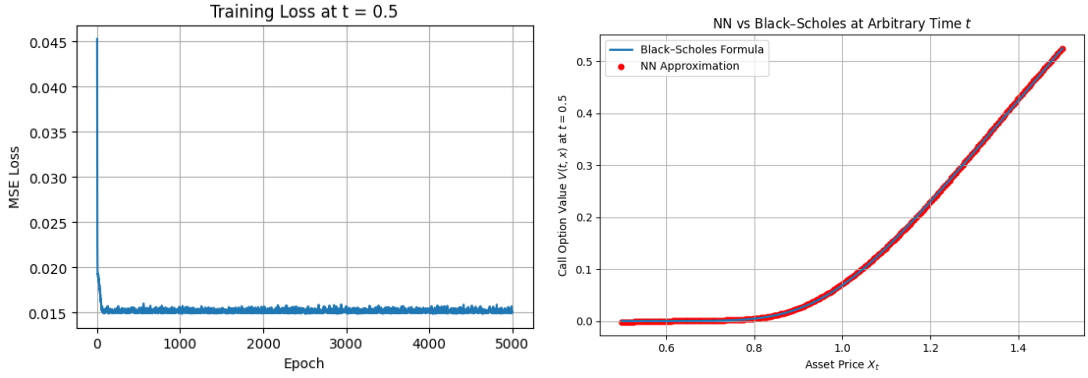


Figure 5: Closed-form simulation results. Left: MSE training loss using exact simulation. Right: Neural network vs. Black–Scholes formula at $t = 0.5$.

**Discussion.** Both implementations demonstrate that the neural network successfully learns the Black–Scholes pricing function at fixed time $t = 0.5$. While both achieve high accuracy, the closed-form simulation slightly outperforms Euler–Maruyama in terms of error. However, this comparison also corroborates

that in the absence of closed-form solutions for PDEs, the Euler–Maruyama scheme would be a viable alternative for data generation while still preserving high levels of accuracy of the model. These results also confirm the solver's validity for pricing European options via Feynman–Kac representations.

### 4.3 Full Surface Approximation of the Black–Scholes Solution

We now train a two-dimensional neural network to approximate the entire Black–Scholes surface $V(t, x)$ over both time and space. This experiment tests the network's ability to learn the full conditional expectation structure encoded by the Feynman–Kac formula, rather than at a fixed time slice.

**Training and Evaluation.** The network is trained over a grid of $(t, x)$ points with the target values generated using the explicit Black–Scholes pricing formula. Figure 6 compares the learned surface $V_{\text{NN}}(t, x)$ to the exact Black–Scholes solution $V_{\text{BS}}(t, x)$, and Figure 7 shows the training loss curve and the approximation quality with the absolute error across the domain.

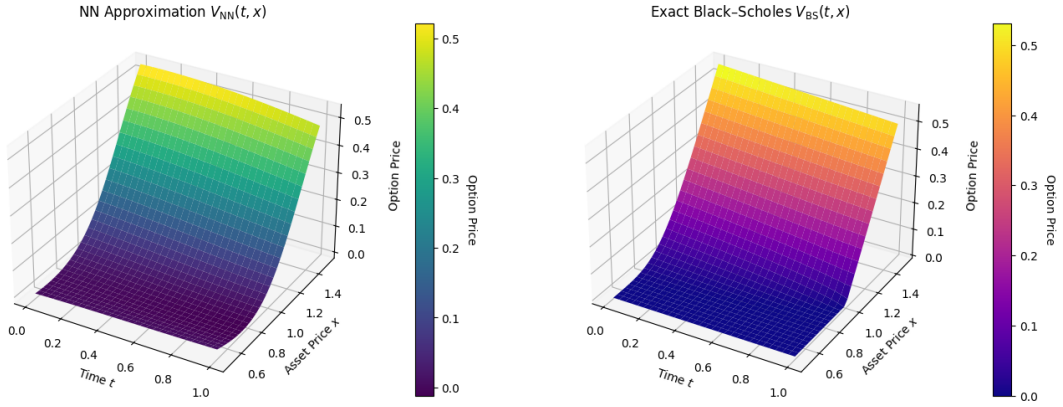


Figure 6: Left: NN approximation $V_{\text{NN}}(t, x)$. Right: Exact Black–Scholes solution $V_{\text{BS}}(t, x)$.

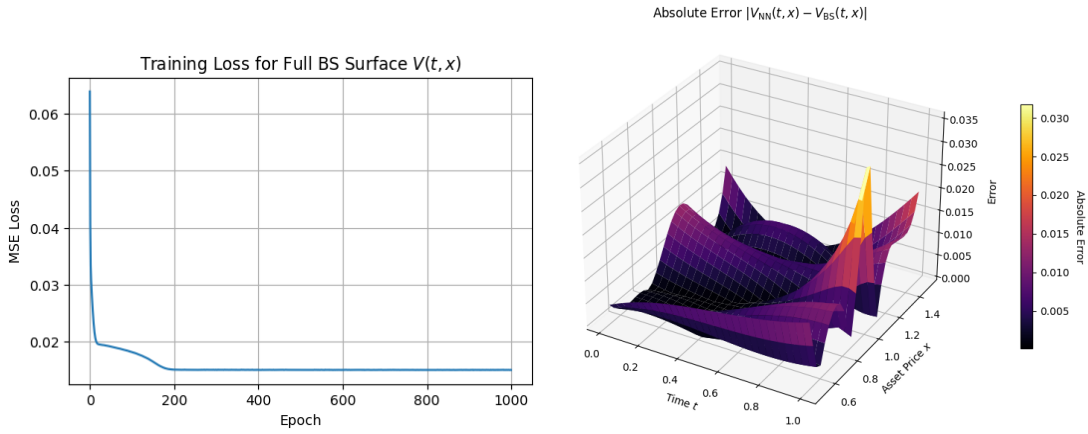


Figure 7: Left: Training loss for 2D neural network approximating $V(t, x)$. Right: Absolute error $|V_{\text{NN}}(t, x) - V_{\text{BS}}(t, x)|$ across the domain.

**Comparison at Fixed Time.** To compare the 2D network against the earlier 1D setup, we extract a slice of the surface at $t = 0.5$. Figures 8 and 9 compare the two networks, as well as compare them against other numerical methods such as finite difference (FD) using the Crank-Nicholson scheme and Monte Carlo (MC).

This implementation shows that a two-dimensional neural network can accurately learn the full Black–Scholes solution surface $V(t, x)$ using samples generated from the analytical formula. Additional experiments with models trained on samples generated using the Euler-Maruyama scheme also showed

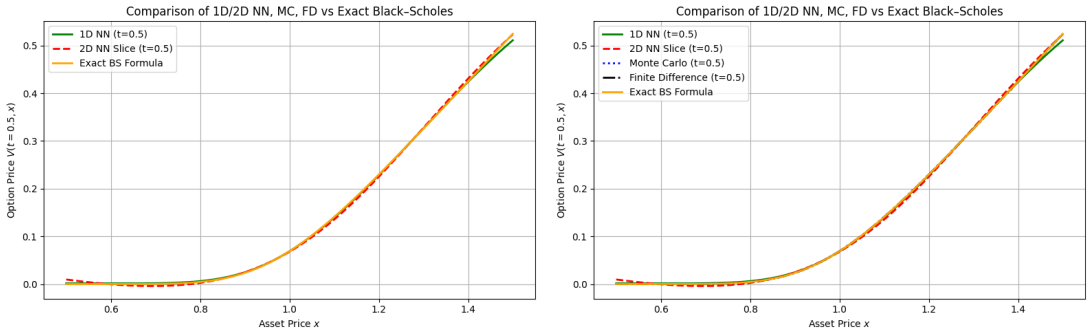


Figure 8: Left: Comparison of 1D and 2D neural network approximations at $t = 0.5$. Right: Comparison of all methods (1D/2D NN, MC, FD, exact) at $t = 0.5$.

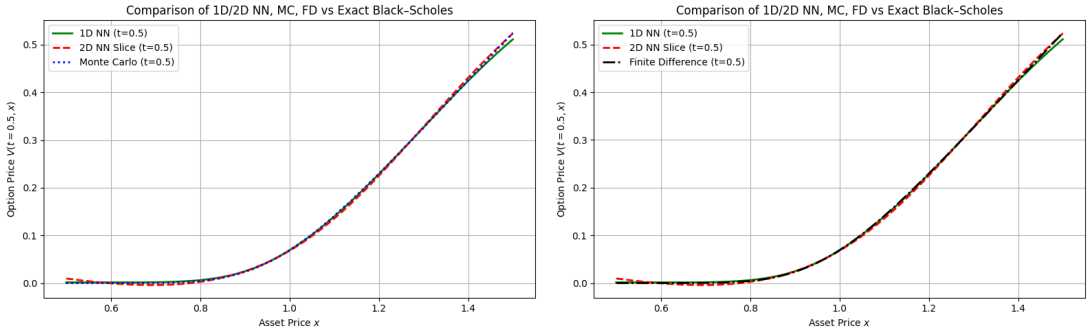


Figure 9: Left: Comparison of 1D and 2D neural networks with the Monte Carlo estimator at $t = 0.5$. Right: Comparison of the same neural networks with the Finite Difference scheme (Crank–Nicolson) at $t = 0.5$. Both approximations show close agreement with the reference numerical methods.

comparative levels of accuracy to models trained on samples generated from the analytical formula. The learned surface matches the exact solution both qualitatively and quantitatively, with absolute errors remaining low across the domain. Furthermore, slices extracted at fixed times show that the 2D network aligns well with the 1D network trained specifically for those time slices, and both agree closely with classical numerical methods such as Monte Carlo and the Crank–Nicolson finite difference scheme. This confirms the viability of neural networks as flexible solvers for linear parabolic PDEs.

# 5 Conclusion

This study demonstrated the viability of combining stochastic simulation with neural network regression to solve linear parabolic PDEs. By leveraging the Feynman–Kac representation, we reformulate the PDE problem as a conditional expectation estimation task, allowing us to employ neural networks trained on simulated data as scalable solvers. The method avoids mesh discretization entirely, offering a practical route to solving PDEs that are otherwise intractable using classical numerical techniques. Through a series of experiments on the heat equation and the Black–Scholes PDE, we verified that both fixed-time and full-surface neural approximations can accurately reproduce known solutions and align with finite difference and Monte Carlo methods.

**Future Work.** Several promising directions remain open. First, this framework should be extended to high-dimensional PDEs, where the curse of dimensionality severely limits traditional grid-based solvers. Second, the empirical success of the two-dimensional network invites theoretical analysis into how neural networks generalize conditional expectations across time and space. Third, a major research avenue lies in adapting this method to *nonlinear* PDEs, potentially by combining the Feynman–Kac representation with tools from stochastic control theory or backward stochastic differential equations (BSDEs). Fourth, integrating variance reduction or importance sampling during the simulation phase could improve the statistical efficiency of training. Lastly, applying this solver to path-dependent or exotic financial derivatives would further demonstrate its flexibility and real-world utility. These extensions highlight the potential of this approach as a general-purpose, probabilistically grounded numerical solver for PDEs.

# Appendix: Results in Stochastic Calculus

## A.1 Brownian Motion

**Definition A.1.1 (Brownian motion).** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. For each $\omega \in \Omega$, suppose there is a continuous function $W(t)$ of $t \geq 0$ that satisfies $W(0) = 0$ and that depends on $\omega$. Then $W(t)$, $t \geq 0$, is called a Brownian motion if for all $0 = t_0 < t_1 < \cdots < t_m$, the increments

$$W(t_1) - W(t_0),\ W(t_2) - W(t_1),\ \ldots,\ W(t_m) - W(t_{m-1})$$

are independent and normally distributed with

$$\mathbb{E}[W(t_{i+1}) - W(t_i)] = 0, \qquad \mathrm{Var}[W(t_{i+1}) - W(t_i)] = t_{i+1} - t_i.$$

**Theorem A.1.2 (Alternative characterizations of Brownian motion).** The following are equivalent:

(i) The increments are independent and normally distributed as above.

(ii) $\{W(t_1), \ldots, W(t_m)\}$ are jointly Gaussian with zero mean and covariance matrix defined by $\mathrm{Cov}(W(t_i), W(t_j)) = \min(t_i, t_j)$.

(iii) $\{W(t_1), \ldots, W(t_m)\}$ have the joint moment-generating function corresponding to (ii).

**Definition A.1.3 (Filtration).** A filtration $\{\mathcal{F}(t)\}_{t \geq 0}$ for Brownian motion $W(t)$ satisfies:

(i) *Information accumulates:* $\mathcal{F}(s) \subseteq \mathcal{F}(t)$ for $0 \leq s < t$.

(ii) *Adaptivity:* $W(t)$ is $\mathcal{F}(t)$-measurable.

(iii) *Independence of future increments:* $W(u) - W(t)$ is independent of $\mathcal{F}(t)$ for $t < u$.

**Theorem A.1.4 (Martingale property).** Brownian motion is a martingale with respect to its natural filtration:

$$\mathbb{E}[W(t) \mid \mathcal{F}(s)] = W(s), \quad 0 \leq s \leq t.$$

**Definition A.1.5 (Quadratic variation).** For a function $f(t)$, the quadratic variation is

$$[f, f](T) = \lim_{\|\Pi\| \to 0} \sum_{j=0}^{n-1} \left[f(t_{j+1}) - f(t_j)\right]^2,$$

where $\Pi = \{t_0, t_1, \ldots, t_n\}$ is a partition of $[0, T]$.
**Theorem A.1.6.** Let $W(t)$ be Brownian motion. Then $[W, W](T) = T$ almost surely.
**Corollary A.1.7.** Informally, we write:

$$dW(t)\, dW(t) = dt, \quad dW(t)\, dt = 0, \quad dt\, dt = 0.$$

**Theorem A.1.8 (Markov property).** Brownian motion is a Markov process:

$$\mathbb{E}[f(W(t)) \mid \mathcal{F}(s)] = g(W(s)),$$

for some Borel-measurable function $g$, whenever $f$ is Borel-measurable and $0 \leq s \leq t$.

## A.2 Stochastic Calculus

**Definition A.2.1 (Itô integral of simple process).** For a simple process $\Delta(t)$, define

$$I(t) = \sum_{j=0}^{k-1} \Delta(t_j)[W(t_{j+1}) - W(t_j)] + \Delta(t_k)[W(t) - W(t_k)],$$

and write

$$I(t) = \int_0^t \Delta(u)\, dW(u).$$

**Theorem A.2.2.** The Itô integral defined above is a martingale.
**Theorem A.2.3 (Itô isometry).**

$$\mathbb{E}[I^2(t)] = \mathbb{E}\left[\int_0^t \Delta^2(u)\,du\right].$$

**Theorem A.2.4 (Quadratic variation of Itô integral).**

$$[I, I](t) = \int_0^t \Delta^2(u)\,du.$$

**Theorem A.2.5 (Properties of the Itô integral).** Let $I(t) = \int_0^t \Delta(u)\,dW(u)$. Then:

(i) Continuity: paths of $I(t)$ are continuous.

(ii) Adaptivity: $I(t)$ is $\mathcal{F}(t)$-measurable.

(iii) Linearity: $I(t) + J(t) = \int_0^t [\Delta(u) \pm \Gamma(u)]\,dW(u)$.

(iv) Martingale property: $I(t)$ is a martingale.

(v) Itô isometry: as stated above.

(vi) Quadratic variation: as above.

**Theorem A.2.6 (Itô–Doeblin formula for Brownian motion).** Let $f(t, x)$ have continuous derivatives $f_t, f_x, f_{xx}$. Then:

$$f(T, W(T)) = f(0, W(0)) + \int_0^T f_t(t, W(t))\,dt + \int_0^T f_x(t, W(t))\,dW(t) + \frac{1}{2}\int_0^T f_{xx}(t, W(t))\,dt.$$

in differential form:

$$df(t, W(t)) = f_t\,dt + f_x\,dW(t) + \frac{1}{2} f_{xx}\,dt.$$

**Definition A.2.7 (Itô process).** A process $X(t)$ is Itô if

$$X(t) = X(0) + \int_0^t \Delta(u)\,dW(u) + \int_0^t \Theta(u)\,du,$$

with $\Delta, \Theta$ adapted.
**Lemma A.2.8.** The quadratic variation of an Itô process is:

$$[X, X](t) = \int_0^t \Delta^2(u)\,du.$$

**Definition A.2.9 (Integral w.r.t. Itô process).**

$$\int_0^t \Gamma(u)\,dX(u) = \int_0^t \Gamma(u)\Delta(u)\,dW(u) + \int_0^t \Gamma(u)\Theta(u)\,du.$$

**Theorem A.2.11 (Itô integral of deterministic integrands).** Let $\Delta(t)$ be deterministic. Then:

$$I(t) = \int_0^t \Delta(s)\,dW(s)$$

is normal with mean 0 and variance $\int_0^t \Delta^2(s)\,ds$.

### A.3 Multivariable Itô Calculus

**Definition A.3.1 (Multidimensional Brownian motion).** A process $(W_1(t), \dots, W_d(t))$ is a $d$-dimensional Brownian motion if:

1. Each $W_i(t)$ is a 1D Brownian motion.
2. $W_i$ and $W_j$ are independent for $i \neq j$.

**Theorem A.3.2 (Itô's formula).** Let $X = (X^1, \dots, X^d)$ be a continuous vector semimartingale and $F \in C^2(\mathbb{R}^d; \mathbb{R})$; then $F(X)$ is a continuous semimartingale and we have

$$F(X_t) = F(X_0) + \sum_i \int_0^t \frac{\partial F}{\partial x_i}(X_s)\, dX_s^i + \frac{1}{2} \sum_{i,j} \int_0^t \frac{\partial^2 F}{\partial x_i \partial x_j}(X_s)\, d[X^i, X^j]_s.$$

In differential form:

$$dF(X_t) = \sum_i \frac{\partial F}{\partial x_i}(X_t)\, dX_t^i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 F}{\partial x_i \partial x_j}(X_t)\, d[X^i, X^j]_t.$$

**Corollary A.3.3 (Itô product rule).**

$$d[X(t)Y(t)] = X(t)\, dY(t) + Y(t)\, dX(t) + dX(t)\, dY(t).$$

# References


[1] Christian Beck, Weinan E, and Arnulf Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Proceedings of the National Academy of Sciences*, 116(34):16805–16810, 2019.

[2] Christian Beck, Wolfgang E, and Arnulf Jentzen. Generalization bounds for deep learning approximations of high-dimensional pdes. *Analysis and Applications*, 19(1):1–46, 2021.

[3] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[4] Jan Blechschmidt and Oliver G. Ernst. Three ways to solve partial differential equations with neural networks — a review. *arXiv preprint arXiv:2102.11802*, 2021. Submitted to *Mathematics of Computation*.

[5] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial*. SIAM, 2nd edition, 2000.

[6] Herbert Brunner et al. *High-dimensional Partial Differential Equations in Science and Engineering*. Contemporary Mathematics. American Mathematical Society, 2013.

[7] Shashank J. Chowdhury and Dongbin Xiu. Machine learning for pdes: A review. *Journal of Computational Physics*, 429:110010, 2021.

[8] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[9] Lawrence C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, 2nd edition, 2010.

[10] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philipp Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black–scholes pdes. *arXiv preprint arXiv:1809.02362*, 2018.

[11] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[12] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[13] John C. Hull. *Options, Futures, and Other Derivatives*. Pearson Prentice Hall, Upper Saddle River, NJ, 6th ed., pearson international edition edition, 2006.

[14] Ioannis Karatzas and Steven E. Shreve. *Brownian Motion and Stochastic Calculus*, volume 113 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1991.

[15] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *arXiv preprint arXiv:1707.03351*, 2017.

[16] Hans Petter Langtangen and Aslak Tveito. *Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*. Springer, 2003.

[17] K.W. Morton and D.F. Mayers. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press, 2nd edition, 2005.

[18] Andrea Pascucci. *PDE and Martingale Methods in Option Pricing*, volume 1971 of *Lecture Notes in Mathematics*. Springer, Berlin, Heidelberg, 2011.

[19] Huyên Pham. *Continuous-time Stochastic Control and Optimization with Financial Applications*, volume 61 of *Stochastic Modelling and Applied Probability*. Springer, Berlin, Heidelberg, 2009.

[20] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[21] Daniel Revuz and Marc Yor. *Continuous Martingales and Brownian Motion*, volume 293 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, Heidelberg, 3rd edition, 1999.

[22] Steven E. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models*. Graduate Texts in Mathematics. Springer, 2004.

[23] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.