Communications in Combinatorics and Optimization
Vol. xx, No. x (xxxx), pp. 1-36
https://doi.org/10.22049/cco.2026.31036.2723

CCO
Commun. Comb. Optim.

Research Article

A hybrid model for portfolio optimization: integrating the machine learning and the cardinality constrained mean-variance approaches

Panumart Sawangtong[1,2,3,*], Alireza Najafi[4]

[1]Department of Mathematics, Faculty of Applied Science,
King Mongkut's University of Technology North Bangkok, Bangkok, Thailand

[2]Research group for fractional calculus theory and applications, Science and Technology Research Institute, King Mongkut's University of Technology North Bangkok, Bangkok, Thailand

[3]Centre of Excellence in Mathematics, MHESI, Bangkok, 10400, Thailand
*panumart.s@sci.kmutnb.ac.th

[4]Department of Applied Mathematics, Faculty of Mathematical Sciences,
University of Guilan, Guilan, Iran
a.r.najafi97@gmail.com

Received: 24 September 2025; Accepted: 20 May 2026
Published Online: 28 May 2026

Abstract: This paper focuses on developing an optimal investment portfolio that combines stocks and related options to minimize unsystematic risk. To achieve this goal, we utilize a deep learning algorithm known as the Deep Galerkin Method (DGM), which accurately prices European call and put options within a long-memory stochastic local volatility framework, thereby enhancing pricing accuracy. Using historical data from 79 stocks in the Russell 2000 Index between 2017 and 2022, we apply various machine learning methods including Random Forest, K-Nearest Neighbors (KNN), and Support Vector Machines (SVM) to identify optimal portfolios suitable for both risk-seeking and risk-averse investors in 2023. To further refine our investment strategy, we integrate the most effective machine learning methods with a Cardinality-Constrained Mean Variance (CCMV) portfolio optimization model. This integration enables us to construct portfolios tailored to different levels of risk tolerance among investors.

Keywords: Deep learning, partial differential equation, option pricing, stochastic local volatility model, portfolio optimization.

AMS Subject classification: 65Nxx, 91G10, 68T07

* *Corresponding Author*

1. Introduction

The stochastic-local volatility (SLV) model emerges as a powerful alternative in the landscape of option pricing and stock price prediction, bridging the gap between local volatility (LV) and stochastic volatility (SV) models. Unlike traditional LV models, which assume that volatility is a deterministic function of the underlying asset price, the SLV model incorporates both local volatility dynamics and stochastic components. This duality allows it to capture more complex market behaviors, such as changing volatility over time and across different levels of the underlying asset price, thus providing a more nuanced view of price movements and risks. One of the critical benefits of the SLV model is its ability to accurately replicate the observed pricing behavior of options across various maturities and strikes, which is often a challenge for purely LV or SV models. By utilizing a stochastic process to describe the volatility, the SLV model can adapt to sudden shifts in market conditions, such as those prompted by economic events or market shocks. This adaptability results in a more realistic portrayal of implied volatility surfaces, allowing traders and risk managers to make better-informed decisions regarding options pricing and hedging strategies. Furthermore, the importance of the SLV model extends beyond theoretical constructs; it provides practical advantages in risk management and investment strategies. For market participants, the ability to predict stock prices and option pricing with greater accuracy directly translates to more effective trading strategies and risk mitigation techniques. As market dynamics become increasingly complex, the SLV model equips practitioners with a robust framework to analyze and navigate the challenges associated with volatility modeling, ensuring that they can better understand and respond to market fluctuations in a timely manner. This makes the SLV model a valuable tool in the arsenal of traders and analysts alike.

In recent years, researchers have studied stochastic-local volatility financial models and calculated different options prices under the models. Guoping Xu and Harry Zheng presented a jump-diffusion model for valuing basket options, where asset prices follow correlated local volatility processes with systematic jumps. They derived a forward partial integral differential equation and applied an asymptotic expansion method, demonstrating that their approach is faster and more accurate than traditional Monte Carlo methods in numerical tests [26]. Stefano Pagliarani and Andrea Pascucci introduced new approximation formulas for local stochastic volatility models that can incorporate Lévy jumps. By expanding the characteristic function in Fourier space, they developed efficient pricing formulas for plain vanilla options and demonstrate their accuracy with numerical results using real market data [19]. Kenichiro Shiraya and Akihiko Takahashi proposed a novel approximation formula for pricing basket options within a local-stochastic volatility model that incorporates jumps in both asset price and volatility processes [22]. Yuri F. Saporito et al. studied the calibration of the Stochastic Local-Volatility (SLV) model, integrating local and stochastic volatility characteristics [21]. Jingtang Ma et al. proved the second-order convergence rates of the continuous-time Markov chain (CTMC) method for option

pricing under stochastic local volatility (SLV) models, including the Heston and SABR models [17]. Yong He et al. examined an optimal investment strategy within a Heston local-stochastic volatility (HLSV) model, focusing on volatility dynamics characterized by slow and fast varying processes [9]. Hyun-Gyoon Kim and Jeong-Hoon Kim introduced a mixed model that integrates local volatility, pure stochastic volatility, and Lévy-type jumps, with volatility and jump intensity modeled as functions of varying stochastic processes [15]. Donghyun Kim et al. presented a novel hybrid SLV model to address the limitations of traditional local and stochastic volatility models in capturing the dynamics of implied volatilities for equity options [14]. In the mentioned researches, the use of the SLV financial models compared to other stochastic stochastic volatility models has increased efficiency. But these researches are based on the the financial model without memory. Using financial models without long memory to predict asset prices can overlook important market dynamics, such as persistent volatility and correlations over time. This simplification may lead to inaccurate forecasts, especially in markets where past events significantly influence future price movements. In the paper, we add the long memory to the SLV financial model and obtain the European option price under the model.

In the past few years, a variety of methods have emerged to address the dimensionality issue that mesh-based methods encounter by incorporating machine learning techniques. Applying a deep neural network to characterize unknown functions is a common component of these methods. For instance, the research conducted by Han and Jentzen [7, 8] employs a deep BSDE method. Using a Feynman–Kac formula, this method changes the nonlinear PDE into a backward stochastic differential equation (BSDE). It then uses a neural network to estimate the gradient of the unknown function. Beck et al. [3] and Huré et al. [12] offer an expansion of this strategy. In order to solve PDEs, the Deep Galerkin Method (DGM) outlined by Sirignano and Spiliopoulos [24] relies on a deep neural network representation of the desired function. To train the network, we minimize losses associated with the differential operator on the function, as well as any initial, terminal, and/or boundary conditions that the solution must satisfy. To obtain the training data for the neural network, we employ a random sampling from the domain containing the specified PDE. A key characteristic of this method is that, in contrast to other prevalent numerical techniques like finite difference methods, it is devoid of a mesh. Research indicates that when applied to high-dimensional PDEs, the DGM may experience less impact from the curse of dimensionality than other numerical approaches [24]. Relatedly, for a certain class of nonlinear PDEs, the research of Hutzenthaler et al. [13] establishes that techniques based on deep learning are able to circumvent the curse of dimensionality when numerically approximating solutions. Furthermore, in the framework of optimal stochastic control and mean field games, Al-Aradi et al. [1] extended the DGM to solve several PDEs.

The first main goal of this paper is to develop a more accurate method for pricing European call and put options. We introduce a new financial model called a long-memory SLV model. Unlike older models, this approach accounts for past price

movements over longer time periods by incorporating the Hurst parameter H (where $3/4 < H < 1$), and it combines both local and random volatility dynamics. To solve the resulting complex partial differential equation (PDE), we employ a deep learning algorithm known as the Deep Galerkin Method (DGM). Through numerical experiments, we demonstrate that the DGM produces reliable option prices that match traditional Monte Carlo simulations while being computationally more efficient, thus providing a practical tool for fast and accurate option pricing.

The second main goal is to use these improved option prices to construct optimal investment portfolios that minimize unsystematic risk. We integrate machine learning methods specifically Random Forest, KNN and SVM with the CCMV optimization model. Using historical daily price data from 79 stocks in the Russell 2000 Index between 2017 and 2022, the machine learning classifiers identify stocks suitable for risk-averse or risk-seeking investors. The most effective classifiers are then combined with the CCMV model to build tailored portfolios for the year 2023. This hybrid framework allows investors to select portfolios that balance risk and return according to their individual tolerance levels, supported by more accurate option prices derived from the long-memory SLV model.

2. Option price PDE

The SLV model serves as a mathematical framework for describing the evolution of an asset price and its associated volatility. To appreciate its role, it is helpful to understand its two main precursors. The local volatility model assumes that volatility is a deterministic function of the asset price and time, which allows it to fit the implied volatility surface well but often fails to capture the random, evolving nature of volatility observed in real markets. In contrast, *stochastic volatility* models introduce a separate random process for volatility, capturing phenomena like volatility clustering but sometimes struggling to exactly replicate all market option prices simultaneously. The model bridges these two approaches by combining a local volatility component with a stochastic one. In the paper, this hybrid nature is extended further by incorporating *long memory*, a feature governed by the Hurst parameter H (with $3/4 < H < 1$). Long memory implies that past price movements and volatility shocks have persistent effects that decay slowly over time, a property observed in many financial time series but absent in standard models. The system of stochastic differential equations given in the paper describes how the asset price S_t and its volatility-related process V_t evolve together. Notably, the volatility driving both processes involves fractional Brownian motion, which accounts for the long-range dependence, while the correlation term captures the leverage effect the tendency for volatility to rise as asset

prices fall. The structure of the long memory version of the SLV model is as follows:

$$\begin{aligned} dS_t &= rS_t dt + \left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right) S_t dM_t^H \\ dV_t &= \frac{1}{\xi}\left(\alpha - V_t\right) dt + \frac{\sigma}{\sqrt{\xi}}\sqrt{V_t} dQ_t^H \\ dM_t^H\ dQ_t^H &= \rho(1 + Ht^{2H-1})dt. \end{aligned} \tag{2.1}$$

In this formulation, r represents the risk-free interest rate, while α is the long-term mean of the volatility level, providing a baseline around which volatility oscillates. The parameter σ denotes the volatility of the volatility, reflecting the degree of variability in the stochastic process of the volatility itself. A key aspect of the model is the Hurst parameter H, which governs the long-range dependence properties of the increments of the stochastic processes $\left\{M_t^H\right\}_{t\geq 0}$ and $\left\{Q_t^H\right\}_{t\geq 0}$. The parameters δ and ξ are constants which adjust the contributions of the stochastic components to the overall dynamics of asset prices and volatility.

The key mathematical challenge is that, in this extended SLV model, there is no closed-form solution for the price of a European option. The paper therefore derives a partial differential equation (PDE) that the option price $C(t, S_t, V_t)$ must satisfy. This derivation follows a classic no-arbitrage approach: one constructs a risk-free portfolio consisting of the option, a quantity of the underlying asset, and a quantity of a volatility-linked asset. By carefully choosing the quantities to eliminate the random (stochastic) components, the resulting portfolio becomes deterministic and must earn the risk-free rate. This procedure transforms the probabilistic problem into a deterministic PDE that depends on time t, the asset price S, and the volatility state V. The PDE is parabolic but highly nonlinear and three-dimensional, making traditional numerical methods (like finite differences) computationally expensive or unstable, particularly because of the long-memory terms and the mixed derivative.

Lemma 1. *The option price $C = \mathcal{C}(t, S_t, V_t)$ is the solution of the following PDE:*

$$\begin{aligned} &\frac{\partial \mathcal{C}}{\partial t} + rS_t\frac{\partial \mathcal{C}}{\partial S_t} + rV_t\frac{\partial \mathcal{C}}{\partial V_t} + \left(\frac{1}{2} + Ht^{2H-1}\right)\left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right)^2 S_t^2\frac{\partial^2 \mathcal{C}}{\partial S_t^2} \\ &+ \left(\frac{1}{2} + Ht^{2H-1}\right)\frac{\sigma^2}{\xi}V_t\frac{\partial^2 \mathcal{C}}{\partial V_t^2} \\ &+ \left(1 + 2Ht^{2H-1}\right)\frac{\sigma\rho}{\sqrt{\xi}}\left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right)\sqrt{V_t}S_t\frac{\partial^2 \mathcal{C}}{\partial S_t \partial V_t} - rC = 0, \end{aligned} \tag{2.2}$$

where $\frac{3}{4} < H < 1$.

Proof. Let us begin by presenting the pay-off functions for the call and put options, defined as follows:

$$\mathcal{C}(T, S, V) = \max\left\{S_T - K, 0\right\}, \tag{2.3}$$

and

$$P(T,S,V)=\max\left\{K-S_T,0\right\}, \tag{2.4}$$

for any $t \geq 0$. In this framework, consider a portfolio X consisting of an option $\mathcal{C}(t,S_t,V_t)$, Φ shares of a volatility asset V, and Υ shares of the underlying asset S. Consequently, the value of the portfolio at time t can be expressed as:

$$X_t=\mathcal{C}(t,S_t,V_t)-\Phi_t V_t-\Upsilon_t S_t. \tag{2.5}$$

Following a time interval δt, the change in the portfolio's value δX_t can be represented as:

$$\delta X_t=\delta\mathcal{C}(t,S_t,V_t)-\Phi_t\delta V_t-\Upsilon_t\delta S_t, \tag{2.6}$$

where δC, δS_t, and δV_t represent the changes in the option and asset prices, and volatility, respectively. Utilizing the fractional Itô lemma for the mixed fractional Brownian motion [18], we can express the differential form of the portfolio in the following manner:

$$\begin{aligned}
dX(t)=&\Big[\frac{\partial\mathcal{C}}{\partial t}+rS_t\frac{\partial\mathcal{C}}{\partial S_t}+\frac{1}{\xi}\left(\alpha-V_t\right)\frac{\partial\mathcal{C}}{\partial V_t}+\left(\frac{1}{2}+Ht^{2H-1}\right)\left(e^{V_t}+\sqrt{\delta}\left(\log(S_t)\right)^2\right)^2 S_t^2\frac{\partial^2\mathcal{C}}{\partial {S_t}^2}\\
&+\left(\frac{1}{2}+Ht^{2H-1}\right)\frac{\sigma^2}{\xi}V_t\frac{\partial^2\mathcal{C}}{\partial {V_t}^2}+\left(1+2Ht^{2H-1}\right)\frac{\sigma\rho}{\sqrt{\xi}}\left(e^{V_t}+\sqrt{\delta}\left(\log(S_t)\right)^2\right)\sqrt{V_t}S_t\frac{\partial^2\mathcal{C}}{\partial S_t\partial V_t}\\
&-rS_t\Upsilon_t-\left(\kappa-\theta V_t\right)\Phi_t\Big]dt\\
&+\left(e^{V_t}+\sqrt{\delta}\left(\log(S_t)\right)^2\right)S_t\left(\frac{\partial\mathcal{C}}{\partial S_t}-\Upsilon_t\right)dM_t^H\\
&+\frac{\sigma}{\sqrt{\xi}}\sqrt{V_t}\left(\frac{\partial\mathcal{C}}{\partial V_t}-\Phi_t\right)dQ_t^H,
\end{aligned} \tag{2.7}$$

where dM_t^H and dQ_t^H denote fractional Brownian motion increments. In order to establish the option price partial differential equation (PDE), we must eliminate the stochastic components from Equation (2.7). Setting $\Upsilon_t=\frac{\partial\mathcal{C}}{\partial S_t}$ and $\Phi_t=\frac{\partial\mathcal{C}}{\partial V_t}$ yields:

$$\begin{aligned}
dX_t=&\Big[\frac{\partial\mathcal{C}}{\partial t}+\left(\frac{1}{2}+Ht^{2H-1}\right)\left(e^{V_t}+\sqrt{\delta}\left(\log(S_t)\right)^2\right)^2 S_t^2\frac{\partial^2\mathcal{C}}{\partial {S_t}^2}+\left(\frac{1}{2}+Ht^{2H-1}\right)\frac{\sigma^2}{\xi}V_t\frac{\partial^2\mathcal{C}}{\partial {V_t}^2}\\
&+\left(1+2Ht^{2H-1}\right)\frac{\sigma\rho}{\sqrt{\xi}}\left(e^{V_t}+\sqrt{\delta}\left(\log(S_t)\right)^2\right)\sqrt{V_t}S_t\frac{\partial^2\mathcal{C}}{\partial S_t\partial V_t}\Big]dt.
\end{aligned} \tag{2.8}$$

When evaluating the return on a risk-free portfolio, we observe that it is expected to match the risk-free interest rate, denoted as r. This equivalence is determined by several factors including arbitrage principles, the risk-free nature of the portfolio, and the opportunity cost associated with investing in alternative assets. Consequently,

we can express this relationship as follows:

$$dX_t = rX_t dt = r\left(\mathcal{C}_t - \Upsilon_t S_t - \Phi_t V_t\right) dt = r\left(\mathcal{C}_t - \frac{\partial \mathcal{C}}{\partial S_t} S_t - \frac{\partial \mathcal{C}}{\partial V_t} V_t\right) dt. \tag{2.9}$$

Consequently, we establish that:

$$\begin{aligned}
&\Big[\frac{\partial \mathcal{C}}{\partial t} + \left(\frac{1}{2} + Ht^{2H-1}\right)\left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right)^2 S_t^2 \frac{\partial^2 \mathcal{C}}{\partial {S_t}^2} + \left(\frac{1}{2} + Ht^{2H-1}\right)\frac{\sigma^2}{\xi} V_t \frac{\partial^2 \mathcal{C}}{\partial {V_t}^2} \\
&\quad + \left(1 + 2Ht^{2H-1}\right)\frac{\sigma\rho}{\sqrt{\xi}}\left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right)\sqrt{V_t} S_t \frac{\partial^2 \mathcal{C}}{\partial S_t \partial V_t}\Big] dt \\
&\quad = r\left(\mathcal{C}_t - \frac{\partial \mathcal{C}}{\partial S_t} S_t - \frac{\partial \mathcal{C}}{\partial V_t} V_t\right) dt.
\end{aligned} \tag{2.10}$$

Therefore, we can conclude that the final version of the PDE governing the option price is given by:

$$\begin{aligned}
&\frac{\partial \mathcal{C}}{\partial t} + rS_t \frac{\partial \mathcal{C}}{\partial S_t} + rV_t \frac{\partial \mathcal{C}}{\partial V_t} + \left(\frac{1}{2} + Ht^{2H-1}\right)\left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right)^2 S_t^2 \frac{\partial^2 \mathcal{C}}{\partial {S_t}^2} \\
&+ \left(\frac{1}{2} + Ht^{2H-1}\right)\frac{\sigma^2}{\xi} V_t \frac{\partial^2 \mathcal{C}}{\partial {V_t}^2} \\
&+ \left(1 + 2Ht^{2H-1}\right)\frac{\sigma\rho}{\sqrt{\xi}}\left(e^{V_t} + \sqrt{\delta}\left(\log(S_t)\right)^2\right)\sqrt{V_t} S_t \frac{\partial^2 \mathcal{C}}{\partial S_t \partial V_t} - r\mathcal{C} = 0.
\end{aligned} \tag{2.11}$$

□

3. The deep Galerkin method

Since the obtained PDE does not have closed form solution, thus we use the DGM to solve this equation. The connection between the SLV model and the DGM solver is therefore deeply complementary. The SLV model provides the complex, realistic, and analytically intractable PDE that captures long-memory, leverage, and stochastic-local effects. The DGM provides a flexible, mesh-free, and scalable numerical technique capable of handling the high-dimensionality (here two spatial variables S and V plus time) and the complicated operator structure while mitigating the curse of dimensionality. As shown in the paper, the DGM approximates the PDE residual, the boundary error, and the terminal error simultaneously. The convergence analysis outlined in the paper demonstrates that, under appropriate regularity conditions, minimizing the loss function leads to a sequence of neural network approximations that converge to the true solution of the PDE. In essence, the SLV model defines what needs to be solved, and the DGM provides a data-driven, trainable how. This synergy is particularly valuable for practitioners because once the network is trained, option

prices can be evaluated almost instantaneously for any given (t, S, V), enabling fast pricing and hedging in real time.

Let $\mathcal{C}(t, y)$ represent the option price function, where $t \in [0, T]$ and space variable $y \in \mathcal{D} \subset \mathbb{R}^d$. We describe the PDE of interest as follows:

$$\begin{cases} \frac{\partial \mathcal{C}}{\partial t}(t, y) + \mathcal{L}\mathcal{C}(t, y) = 0, & (t, y) \in [0, T] \times \mathcal{D}, \\ \mathcal{C}(t, y) = f(t, y), & (t, y) \in [0, T] \times \partial\mathcal{D}, \\ \mathcal{C}(T, y) = \mathcal{C}_T(y), & y \in \mathcal{D}, \end{cases} \tag{3.1}$$

where $\mathcal{L}$ is a operator. Our goal is to utilize an approximating function $\tilde{\mathcal{C}}(t, y; \Pi)$, defined by a deep neural network with parameter set Π, to approximate $\mathcal{C}$. The loss function for training the neural network combines three terms: the errors from the operator, terminal condition, and boundary can be defined. Thus, we can express the loss function as follows:

$$\begin{aligned} \mathcal{J}(\tilde{\mathcal{C}}) = &\left\| \frac{\partial \tilde{\mathcal{C}}}{\partial t}(t, y; \Pi) + \mathcal{L}\tilde{\mathcal{C}}(t, y; \Pi) \right\|^2_{[0,T]\times\mathcal{D},\vartheta_1} + \|\tilde{\mathcal{C}}(t, y; \Pi) - f(t, y)\|^2_{[0,T]\times\partial\mathcal{D},\vartheta_2} \\ &+ \|\tilde{\mathcal{C}}(T, y; \Pi) - \mathcal{C}_T(y)\|^2_{\mathcal{D},\vartheta_3}, \end{aligned}$$

The error is quantified in terms of the L^2 norm in all of the aforementioned expressions, i.e. $\|\tilde{\mathcal{C}}(y)\|^2_{\gamma,\vartheta} = \int_\gamma |\tilde{\mathcal{C}}(y)|^2 \vartheta(y) dy$, where $\vartheta(y)$ is a density specified across the region γ. Next, we implement stochastic gradient descent to reduce the loss function. To be more precise, we implement the Algorithm 1.

Algorithm 1 DGM algorithm

1. Set up the parameter vector Π and the step size α_n.
2. Make a random selection of points from the domains $[0, T] \times \mathcal{D}$ and $[0, T] \times \partial\mathcal{D}$:
   From $[0, T] \times \mathcal{D}$, uniformly generate (t_n, y_n).
   From $[0, T] \times \partial\mathcal{D}$, uniformly generate (t_n^*, y_n^*).
   From $\mathcal{D}$, uniformly generate y_n^{**}.
3. Calculate the loss function $\mathcal{J}(P_n, \Pi_n)$ at P_n, where $P_n = \{(t_n, y_n), (t_n^*, y_n^*), y_n^{**}\}$ is the set of randomly selected points from previous step:

$$\begin{aligned} \mathcal{J}(P_n, \Pi_n) = &\left(\frac{\partial \tilde{\mathcal{C}}}{\partial t}(t_n, y_n; \Pi_n) + \mathcal{L}\tilde{\mathcal{C}}(t_n, y_n; \Pi_n) \right)^2 \\ &+ \left(\tilde{\mathcal{C}}(t_n^*, y_n^*; \Pi_n) - f(t_n^*, y_n^*) \right)^2 \\ &+ \left(\tilde{\mathcal{C}}(T, y_n^{**}; \Pi_n) - \mathcal{C}_T(y_n^{**}) \right)^2, \end{aligned}$$

4. Perform the following updates to the parameter set Π:

$$\Pi_{n+1} = \Pi_n - \alpha_n \nabla_\Pi \mathcal{J}(P_n, \Pi_n),$$

4. Perform steps 2-4 until $\|\Pi_{n+1} - \Pi_n\|$ is small.

Sirignano and Spiliopoulos [24] developed an architecture that is similar to the LSTM and Highway networks described in [10] and [25], respectively. This architecture consists of three levels, known as DGM layers: an input layer, a hidden layer, and an output layer. Moreover, the structure can be easily extended to incorporate additional hidden layers. Typically, each DGM layer receives as inputs both the output from the previous DGM layer and the initial mini-batch inputs $\vec{y}$, which in this instance represent a collection of randomly sampled time-space points. Following this process, the architecture produces a vector-valued output denoted as $\tilde{\mathcal{C}}$, which provides the neural network's approximation of the target function $\mathcal{C}$ at the mini-batch points.
In a DGM layer, mini-batch inputs and the output from the preceding layer undergo a sequence of operations that are similar to those utilised in highway networks. The architecture is represented by the following equations:

$$\begin{aligned} S^1 &= \phi(W^1\vec{y} + b^1), \\ Z^l &= \phi(U^{z,l}\vec{y} + W^{z,l}S^l + b^{z,l}), \\ G^l &= \phi(U^{g,l}\vec{y} + W^{g,l}S^l + b^{g,l}), \\ R^l &= \phi(U^{r,l}\vec{y} + W^{r,l}S^l + b^{r,l}), \\ H^l &= \phi(U^{h,l}\vec{y} + W^{h,l}(S^l \odot R^l) + b^{h,l}), \\ S^{l+1} &= (1 - G^l) \odot H^l + Z^l \odot S^l, \\ \tilde{\mathcal{C}}(t, y; \Pi) &= WS^{L+1} + b, \end{aligned}$$

Where $l = 1, ..., L$. L stands for the total number of layers. The added vector of the sampled time-space data is denoted by $\vec{y} = (t, y)$, while $\odot$ is the element-wise multiplication. We denote the number of units in each layer by M. Moreover, $\phi : \mathbb{R}^M \to \mathbb{R}^M$ is an activation function. Weight matrices and bias vector parameters have the following dimensions:

$$\begin{aligned} &W^1 \in \mathbb{R}^{M\times(d+1)}, b^1 \in \mathbb{R}^M, \\ &\quad U^{j,l} \in \mathbb{R}^{M\times(d+1)}, W^{j,l} \in \mathbb{R}^{M\times M}, b^{j,l} \in \mathbb{R}^M, j \in \{z, g, r, h\}, \\ &\qquad W \in \mathbb{R}^{1\times M}, b \in \mathbb{R}. \end{aligned}$$

Π denotes the collection of parameters that are estimated::

$$\Pi = \left\{W^1, b^1, \left(U^{z,l}, W^{z,l}, b^{z,l}\right)_{l=1}^{l=L}, \left(U^{g,l}, W^{g,l}, b^{g,l}\right)_{l=1}^{l=L}, \left(U^{r,l}, W^{r,l}, b^{r,l}\right)_{l=1}^{l=L}, \left(U^{h,l}, W^{h,l}, b^{h,l}\right)_{l=1}^{l=L}, W, b\right\}.$$

Analogous to LSTMs, each layer generates weights depending on the previous layer to decide how much data is passed on to the subsequent layer. As stated in [24], incorporating repeated element-wise multiplication of nonlinear functions aids in capturing the "sharp turn" characteristics inherent in increasingly complex functions.
The DGM network has almost eight times as many parameters in each hidden layer as a typical dense layer, in contrast to a multilayer perceptron (MLP). This is due to the

fact that each DGM layer contains eight weight matrices and four bias vectors, whereas the MLP network contains only one weight matrix and one bias vector, assuming comparable matrix and vector dimensions. Furthermore, the LSTM-inspired design of DGM networks effectively addresses the vanishing gradient problem, a common challenge encountered by deep MLPs, while also being sufficiently flexible to model complex functions. It is important to mention that, in each iteration, the initial input is included in the calculations of all intermediary steps, thereby minimizing the likelihood of vanishing gradients affecting the output function concerning $\vec{y}$. Sirignano and Spiliopoulos [24] highlighted that the architecture of a neural network plays a crucial role in its efficacy, and that well-informed architectural choices that leverage knowledge from the specific application can significantly enhance performance. In the research, we maintain the same architectural framework for the neural network.

3.1. Deep Galerkin method convergence

We define $\mathcal{S}^n$ as the set of neural networks with a single hidden layer and n hidden units as:

$$\mathcal{S}^n(\phi) = \left\{ \tilde{\mathcal{C}}(t,S,V) : \mathbb{R}^3 \to \mathbb{R} | \tilde{\mathcal{C}}(t,S,V) = \sum_{i=1}^{n} \chi_i \phi(\eta_{i,1} t + \eta_{i,2} S + \eta_{i,3} V + d_i) \right\} \quad (3.2)$$

where ϕ represents the activation function that it is bounded and non-constant. Also, $\chi_i, \eta_{i,j}, d_i \in \mathbb{R}$ are the network parameters. Let $\tilde{\mathcal{C}}^n$ denote a neural network with n hidden units that minimizes the loss function $\mathcal{J}(\tilde{\mathcal{C}})$, specifically designed for the European option pricing PDE. Under certain conditions, we show that:

1. There exists $\tilde{\mathcal{C}}^n \in \mathcal{S}^n$ such that $\mathcal{J}(\tilde{\mathcal{C}}) \to 0$ as $n \to \infty$,

2. The sequence $\{\tilde{\mathcal{C}}^n\}_{n=1}^{\infty}$ converges strongly to the PDE solution $\mathcal{C}$ in $L^q([0,T]\times\mathcal{D})$ for $q < 2$.

This convergence result ensures that the neural network solution approximates the true PDE solution accurately as the number of hidden units increases. We present the precise theorem and its proof in this section. The proof leverages the approximation capabilities of neural networks alongside the continuity property of the PDE.

3.2. Analysis of L^2 Error Convergence of $\mathcal{J}(\tilde{\mathcal{C}})$

We consider the bounded domain $\mathcal{D} \subset \mathbb{R}^d$ with a smooth boundary $\partial\mathcal{D}$, and define $\mathcal{D}_T = [0,T) \times \mathcal{D}$ and $\partial\mathcal{D}_T = [0,T) \times \partial\mathcal{D}$. We adapt our discussion to the divergence form of the specific parabolic PDE governing the pricing of European options, given by (2.2). For notational simplicity, let us define the operator $\mathfrak{R}$ corresponding to the

PDE. Specifically, let:

$$\begin{cases} \mathfrak{R}[C](t,y) = 0, & (t,y) \in \mathcal{D}_T, \\ \mathcal{C}(T,y) = \mathcal{C}_T(y), & y \in \mathcal{D}, \\ \mathcal{C}(t,y) = f(t,y), & (t,y) \in \partial\mathcal{D}_T, \end{cases} \tag{3.3}$$

where

$$\begin{aligned} \mathfrak{R}[\mathcal{C}](t,y) &= \partial_t \mathcal{C}(t,y) - \operatorname{div}(\mathcal{A}(t,y,\mathcal{C}(t,y),\nabla\mathcal{C}(t,y))) + B(t,y,\mathcal{C}(t,y),\nabla\mathcal{C}(t,y)) \\ &= \partial_t \mathcal{C}(t,y) - \sum_{i,j=1}^{d} \frac{\partial \mathcal{A}_i(t,y,\mathcal{C}(t,y),\nabla\mathcal{C}(t,y))}{\partial \mathcal{C}_{y_j}} \partial_{y_i,y_j}\mathcal{C}(t,y) + \hat{\mathcal{B}}(t,y,\mathcal{C}(t,y),\nabla\mathcal{C}(t,y)), \end{aligned}$$

where

$$\hat{\mathcal{B}}(t,y,\mathcal{C},\nabla\mathcal{C}) = B(t,y,\mathcal{C},\nabla\mathcal{C}) - \sum_{i=1}^{d} \frac{\partial \mathcal{A}_i(t,y,\mathcal{C},\nabla\mathcal{C})}{\partial \mathcal{C}} \partial_{y_i}\mathcal{C} - \sum_{i=1}^{d} \frac{\partial \mathcal{A}_i(t,y,\mathcal{C},\nabla\mathcal{C})}{\partial y_i}$$

As established in classical PDE theory [16], a unique solution $\mathcal{C}(t,y)$ exists for the PDE such that:

$$\sup_{(t,y)\in\mathcal{D}_T} |\nabla_y^{(m)} \mathcal{C}(t,y)| < \infty \quad \text{for} \quad m = 1,2 \tag{3.4}$$

Theorem 1. *Assume The PDE coefficients satisfy smoothness and growth conditions ensuring that nonlinear terms are locally Lipschitz with respect to $\mathcal{C}$ and its gradient, with Lipschitz constants exhibiting at most polynomial growth in $\mathcal{C}$ and $\nabla\mathcal{C}$, uniformly with respect to (t,y). Then, for every ε, there exist a constant $\mathcal{K}$ depending on $\sup_{\mathcal{D}_T}|\mathcal{C}|, \sup_{\mathcal{D}_T}|\nabla_y^{(m)}\mathcal{C}|, m = 1,2$, and a neural network $\tilde{\mathcal{C}} \in \mathcal{S}(\phi) = \cup_{n\geq 1}\mathcal{S}^n(\phi)$ such that:*

$$\mathcal{J}(\tilde{\mathcal{C}}) \leq \mathcal{K}\varepsilon,$$

where $\mathcal{J}(\tilde{\mathcal{C}})$ is the L^2 error measuring the residual of the neural network approximation relative to the PDE and its terminal and boundary conditions.

Proof. According to Theorem 3 in [11], for any $\varepsilon > 0$ and $\mathcal{C} \in \mathbb{C}^{1,2}([0,T]\times\mathbb{R}^d)$ there exists a neural network $\tilde{\mathcal{C}} \in \mathcal{S}(\phi)$ such that

$$\sup_{(t,y)\in\mathcal{D}_T} \left|\partial_t \mathcal{C}(t,y) - \partial_t \tilde{\mathcal{C}}(t,y;\Pi)\right| + \max_{|m|\leq 2} \sup_{(t,y)\in\bar{\mathcal{D}}_T} \left|\partial_x^{(m)} \mathcal{C}(t,y) - \partial_x^{(m)} \tilde{\mathcal{C}}(t,y;\Pi)\right| < \varepsilon \tag{3.5}$$

The mapping $(\rfloor_1,\rfloor_2) \mapsto \hat{\mathcal{B}}(t,y,\rfloor_1,\rfloor_2)$ is locally Lipschitz in $(\rfloor_1,\rfloor_2)$, with a Lipschitz constant that has at most polynomial growth in $\rfloor_1$ and $\rfloor_2$, uniformly with respect to (t,y). Formally, for $0 \leq \varrho_i < \infty, i = 1,\cdots,4$:

$$|\hat{\mathcal{B}}(t,y,\rfloor_1,\rfloor_2)-\hat{\mathcal{B}}(t,y,\rfloor_1^*,\rfloor_2^*)| \leq \left(|\rfloor_1|^{\varrho_1/2}+|\rfloor_2|^{\varrho_2/2}+|\rfloor_1^*|^{\varrho_3/2}+|\rfloor_2^*|^{\varrho_4/2}\right)(|\rfloor_1-\rfloor_1^*|+|\rfloor_2-\rfloor_2^*|).$$

Thus, using Hölder inequality with exponents s_1, s_2, and (3.5), we have

$$\begin{aligned}
&\int_{\mathcal{D}_T}\left|\hat{\mathcal{B}}\left(t,y,\tilde{\mathcal{C}},\nabla_x\tilde{\mathcal{C}}\right)-\hat{\mathcal{B}}\left(t,y,\mathcal{C},\nabla_x C\right)\right|^2 d\vartheta_1(t,y)\\
\leq&\mathcal{K}\left(\int_{\mathcal{D}_T}\left(|\tilde{\mathcal{C}}(t,y;\Pi)-\mathcal{C}(t,y)|^{\varrho_1}+\left|\nabla_x\tilde{\mathcal{C}}(t,y;\Pi)-\nabla_x\mathcal{C}(t,y)\right|^{\varrho_2}+|\mathcal{C}(t,y)|^{r^*}+|\nabla_x\mathcal{C}(t,y)|^{r^{**}}\right)^{s_1}d\vartheta_1(t,y)\right)^{1/s_1}\\
&\times\left(\int_{\mathcal{D}_T}\left(|\tilde{\mathcal{C}}(t,y;\Pi)-\mathcal{C}(t,y)|^2+\left|\nabla_x\tilde{\mathcal{C}}(t,y;\Pi)-\nabla_x\mathcal{C}(t,y)\right|^2\right)^{s_2}d\vartheta_1(t,y)\right)^{1/s_2}\\
\leq&\mathcal{K}\left(\varepsilon^{\varrho_1}+\varepsilon^{\varrho_2}+\sup_{\mathcal{D}_T}|C|^{r^*}+\sup_{\mathcal{D}_T}|\nabla_x C|^{r^{**}}\right)\varepsilon^2 \qquad (3.6)
\end{aligned}$$

where $r^*=\max\{\varrho_1,\varrho_2\}, r^{**}=\max\{\varrho_2,\varrho_4\}$, and $\mathcal{K}<\infty$ is a constant (possibly changing from line to line).
Also, for the differential operator $\partial A_i(t,y,\rfloor,d)/\partial d_j$ we have

$$\left|\frac{\partial\mathcal{A}_i(t,y,\rfloor,d)}{\partial d_j}-\frac{\partial\mathcal{A}_i(t,y,c^*,d^*)}{\partial d_j^*}\right| \leq \left(|c|^{\varrho_1/2}+|d|^{\varrho_2/2}+|c^*|^{\varrho_3/2}+|d^*|^{\varrho_4/2}\right)(|c-c^*|+|d-d^*|).$$

For simplicity, let

$$\zeta(t,y,\mathcal{C},\nabla\mathcal{C},\nabla^2 C)=\sum_{i,j=1}^{d}\frac{\partial\mathcal{A}_i(t,y,\mathcal{C}(t,y),\nabla\mathcal{C}(t,y))}{\partial\mathcal{C}_{x_j}}\partial_{x_i,x_j}\mathcal{C}(t,y)$$

Similar to the process of obtaining (3.6), and using Hölder inequality, we have

$$\int_{\mathcal{D}_T}|\zeta(t,y,\mathcal{C},\nabla\mathcal{C},\nabla^2 C)-\zeta(t,y,\tilde{\mathcal{C}},\nabla\tilde{\mathcal{C}},\nabla^2\tilde{\mathcal{C}})|d\vartheta_1(t,y)\leq\mathcal{K}\varepsilon^2. \qquad (3.7)$$

For the PDE defined in this study, the operator $\mathfrak{R}[C]$ satisfies $\mathfrak{R}[C]=0$, where $\mathcal{C}$ is the solution of PDE. Applying (3.5)-(3.7), allows the objective function to be expressed as:

$$\begin{aligned}
\mathcal{J}(\tilde{\mathcal{C}})&=\|\mathfrak{R}[\tilde{\mathcal{C}}](t,y)\|^2_{\mathcal{D}_T,\vartheta_1}+\|\tilde{\mathcal{C}}(t,y;\Pi)-f(t,y)\|^2_{\partial\mathcal{D}_T,\vartheta_2}+\left\|\tilde{\mathcal{C}}(T,y;\Pi)-\mathcal{C}_T(y)\right\|^2_{\mathcal{D},\vartheta_3}\\
&=\|\mathfrak{R}[\tilde{\mathcal{C}}](t,y)-\mathfrak{R}[C](t,y)\|^2_{\mathcal{D}_T,\vartheta_1}+\|\tilde{\mathcal{C}}(t,y;\Pi)-f(t,y)\|^2_{\partial\mathcal{D}_T,\vartheta_2}+\left\|\tilde{\mathcal{C}}(T,y;\Pi)-\mathcal{C}_T(y)\right\|^2_{\mathcal{D},\vartheta_3}\\
&\leq\int_{\mathcal{D}_T}\left|\partial_t\mathcal{C}(t,y)-\partial_t\tilde{\mathcal{C}}(t,y;\Pi)\right|^2 d\vartheta_1(t,y)+\int_{\mathcal{D}_T}\left|\zeta\left(t,y,\mathcal{C},\nabla\mathcal{C},\nabla^2 C\right)-\zeta\left(t,y,\tilde{\mathcal{C}},\nabla\tilde{\mathcal{C}},\nabla^2\tilde{\mathcal{C}}\right)\right|^2 d\vartheta_1(t,y)\\
&+\int_{\mathcal{D}_T}\left|\hat{\mathcal{B}}\left(t,y,\tilde{\mathcal{C}},\nabla_x\tilde{\mathcal{C}}\right)-\hat{\mathcal{B}}\left(t,y,\mathcal{C},\nabla_x C\right)\right|^2 d\vartheta_1(t,y)+\int_{\partial\mathcal{D}_T}|\tilde{\mathcal{C}}(t,y;\Pi)-\mathcal{C}(t,y)|^2 d\vartheta_2(t,y)+\\
&+\int_{\mathcal{D}}|\tilde{\mathcal{C}}(T,y;\Pi)-\mathcal{C}(T,y)|^2 d\vartheta_3(t,y)\\
&\leq\mathcal{K}\varepsilon^2.
\end{aligned}$$

This final step concludes the proof of the theorem following the rescaling of ε. □

3.3. Neural Network Convergence to the Solution of the PDE

It is important to note that $\mathcal{J}(\tilde{\mathcal{C}}) \to 0$, does not automatically imply $\tilde{\mathcal{C}}^n \to \mathcal{C}$, since the loss function only provides L^2-control over the approximation error. We prove that $\tilde{\mathcal{C}}$ is convergence to the solution $\mathcal{C}$ of the following PDE:

$$\begin{cases} \mathfrak{R}[\mathcal{C}](t,y) = 0, & (t,y) \in \mathcal{D}_T, \\ \mathcal{C}(T,y) = \mathcal{C}_T(y), & y \in \mathcal{D}, \\ \mathcal{C}(t,y) = 0, & (t,y) \in \partial\mathcal{D}_T, \end{cases} \tag{3.8}$$

For mathematical convenience, we restrict our analysis to homogeneous boundary conditions. This simplification allows us to focus on the core properties of the neural network approximation without additional complexity. The loss functional is defined as:

$$\mathcal{J}(\tilde{\mathcal{C}}) = \|\mathfrak{R}[\tilde{\mathcal{C}}]\|^2_{2,\mathcal{D}_T} + \|\tilde{\mathcal{C}}\|^2_{2,\partial\mathcal{D}_T} + \left\|\tilde{\mathcal{C}}(T,\cdot) - \mathcal{C}_T\right\|^2_{2,\mathcal{D}},$$

By Theorem 1, for sufficiently large n, the neural network approximation satisfies:

$$\mathcal{J}(\tilde{\mathcal{C}}^n) \to 0, \text{ as } n \to \infty.$$

The sequence $\{\tilde{\mathcal{C}}^n\}_{n=1}^{\infty}$ satisfies a perturbed version of the original PDE, with a source term $\mathcal{E}^n(t,y)$:

$$\begin{cases} \mathfrak{R}(\tilde{\mathcal{C}}^n)(t,y) = \mathcal{E}^n(t,y), & (t,y) \in \mathcal{D}_T, \\ \tilde{\mathcal{C}}^n(T,y) = \mathcal{C}^n_T(y), & y \in \mathcal{D}, \\ \tilde{\mathcal{C}}^n(t,y) = f^n(t,y), & (t,y) \in \partial\mathcal{D}_T, \end{cases} \tag{3.9}$$

where $\mathcal{E}^n, \mathcal{C}^n_T, f^n$ are approximation errors satisfying:

$$\|\mathcal{E}^n\|^2_{2,\mathcal{D}_T} + \|f^n\|^2_{2,\partial\mathcal{D}_T} + \|\mathcal{C}^n_T - \mathcal{C}_T\|^2_{2,\mathcal{D}} \to 0, \text{ as } n \to \infty. \tag{3.10}$$

Theorem 2. *We assume the following conditions hold:*

1. *There exist a constant $\mathcal{M} > 0$, and non-negative functions $\delta(t,y)$ and $\eta(t,y)$ such that for all $(t,y) \in \mathcal{D}_T$, the following bounds are satisfied:*

$$\|\mathcal{A}(t,y,\rfloor_1,\rfloor_2)\| \le \mathcal{M}(\delta(t,y) + \|\rfloor_2\|), \quad |B(t,y,\rfloor_1,\rfloor_2)| \le \eta(t,y)\|\rfloor_2\|,$$

   where $\delta \in L^2(\mathcal{D}_T)$, and $\eta \in L^{d+2+\nu}(\mathcal{D}_T)$, for $\nu > 0$.

2. *For each $n \in \mathbb{N}, \tilde{\mathcal{C}}^n$ belongs to $C^{1,2}\left(\overline{\mathcal{D}}_T\right)$, and the sequence $\{\tilde{\mathcal{C}}^n\}_{n\in\mathbb{N}} \in L^2(\mathcal{D}_T)$.*

By considering the conditions, the sequence $\{\tilde{\mathcal{C}}^n\}_{n\in\mathbb{N}}$ *converges strongly in* $L^q(\mathcal{D}_T)$ *to* $\mathcal{C}$ *((3.8)) for every* $q < 2$. *Furthermore, if the family* $\left\{\tilde{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$ *is uniformly bounded with respect to* n *and is equicontinuous, then the convergence of* $\tilde{\mathcal{C}}^n$ *to* $\mathcal{C}$ *is uniform over* $\mathcal{D}_T$.

Proof. Results from [20], together with those presented in Chapter V.6 of [16], establish the existence, regularity, and uniqueness for equation (3.8). For bounded domains, the boundedness of the solution can be deduced from Theorem 2.1 of [20], and results in Chapter V.2 of [16]. Let us denote the neural network approximation of the (3.9) solution with $f^n(t,y) = 0$, as $\hat{\mathcal{C}}^n(t,y)$. Using Lemma 4.1 of [20], and assumptions on the growth of $\mathcal{A}(t,y,\rfloor_1,\rfloor_2)$, and $B(t,y,\rfloor_1,\rfloor_2)$, the sequence $\left\{\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$ is uniformly bounded in $L^\infty\left(0,T;L^2(\mathcal{D})\right)\cap L^2\left(0,T;W_0^{1,2}(\mathcal{D})\right)$. These uniform bounds allow us to extract a subsequence, denoted again by $\left\{\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$, which converges to a function $\mathcal{C}$ in the weak-* sense in $L^\infty\left(0,T;L^2(\mathcal{D})\right)$, and weakly in $L^2\left(0,T;W_0^{1,2}(\mathcal{D})\right)$, as well as to a function $\mathcal{C}$ weakly in $L^2(\mathcal{D})$ for each fixed $t\in[0,T)$.
Applying Hölder's inequality with exponents s_1, s_2, for $\rho = 1+\frac{d}{d+4}\in(1,2)$ we have:

$$\begin{aligned}\int_{\mathcal{D}_T}\left|B\left(t,y,\hat{\mathcal{C}}^n,\nabla_y\hat{\mathcal{C}}^n\right)\right|^\rho dtdy &\le \int_{\mathcal{D}_T}|\eta(t,y)|^\rho\left|\nabla_x\hat{\mathcal{C}}^n(t,y)\right|^\rho dtdy\\ &\le\left(\int_{\mathcal{D}_T}|\eta(t,y)|^{s_1\rho}dtdy\right)^{1/s_1}\left(\int_{\mathcal{D}_T}\left|\nabla_y\hat{\mathcal{C}}^n(t,y)\right|^{s_2\rho}dtdy\right)^{1/s_2},\end{aligned}\tag{3.11}$$

By choosing $s_2 = 2/\rho > 1$, we obtain $s_1 = \frac{2}{2-\rho}$. Therefore $s_1\rho = d+2$. From assumption $\eta(t,y)\in L^{d+2}(\mathcal{D}_T)$, and uniform boundedness of $\left\|\nabla_y\hat{\mathcal{C}}^n\right\|_2$, we can deduce that there is a constant $\mathcal{K}<\infty$, such that

$$\int_{\mathcal{D}_T}\left|B\left(t,y,\hat{\mathcal{C}}^n,\nabla_x\hat{\mathcal{C}}^n\right)\right|^\rho dtdy\le\mathcal{K},$$

This estimate, combined with the growth conditions for $\mathcal{A}$ in the theorem assumptions , ensures that $\left\{\partial_t\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$ is uniformly bounded with respect to n in $L^{1+d/(d+4)}(\mathcal{D}_T)$ and $L^2\left(0,T;W^{-1,2}(\mathcal{D})\right)$. We consider the conjugate relationship $1/k_1+1/k_2=1$, where $k_2>\max\{2,d\}$. Using the embeddings $W^{-1,2}(\mathcal{D})\subset W^{-1,k_1}(\mathcal{D})$, $L^\rho(\mathcal{D})\subset W^{-1,k_1}(\mathcal{D})$, and $L^2(\mathcal{D})\subset W^{-1,k_1}(\mathcal{D})$, it follows that $\left\{\partial_t\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$ is uniformly bounded in $L^1\left(0,T;W^{-1,k_1}(\mathcal{D})\right)$. Consider Sobolev spaces $M=W_0^{1,2}(\mathcal{D}), N=L^2(\mathcal{D})$, and $Q=W^{-1,k_1}(\mathcal{D})$, where $M\subset N\subset Q$, and the embedding $M\subset N$ is compact. By Corollary 4 of [23], $\left\{\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$ is relatively compact in $L^2(\mathcal{D}_T)$, implying strong convergence to $\mathcal{C}$ in $\left\{\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$. As a result, $\left\{\hat{\mathcal{C}}^n\right\}_{n\in\mathbb{N}}$ converges to $\mathcal{C}$ almost everywhere in $\mathcal{D}_T$. The nonlinearity of the functions $\mathcal{A}$ and B with respect to the

gradient prevents direct passage to the limit in the weak formulation. However, the uniform boundedness of $\left\{\hat{\mathcal{C}}_n\right\}_{n\in\mathbb{N}}$ in $L^2\left(0,T;W_0^{1,2}(\mathcal{D})\right)$ and its weak convergence to $\mathcal{C}$ in the same space enables us to conclude, as shown in Theorem 3.3 of [4], that

$$\nabla\hat{\mathcal{C}}^n \to \nabla\mathcal{C}, \text{ almost everywhere in } \mathcal{D}_T,$$

We show the nonlinear terms $\mathcal{A}\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)$ and $B\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)$ converge strongly in their respective $L^q\left(\mathcal{D}_T\right)$ and $L^\rho\left(\mathcal{D}_T\right)$ spaces as $n\to\infty$, where $1<q<2$, and $\rho = 1+\frac{d}{d+4}$, using Vitali's theorem. We consider $q<2$, and a measurable set $X\subset\mathcal{D}_T$, the growth assumptions on $\mathcal{A}(t,y,\rfloor_1,\rfloor_2)$, and Hölder's inequality with exponent $2/q$, imply

$$\begin{aligned}
\int_X \left|\mathcal{A}\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)\right|^q dtdy &\le \mathcal{K}\left[\int_X |\delta(t,y)|^q dtdy + \int_X \left|\nabla\hat{\mathcal{C}}^n(t,y)\right|^q dtdy\right]\\
&\le \mathcal{K}\left[\int_X |\delta(t,y)|^q dtdy + \left(\int_{\mathcal{D}_T}\left|\nabla\hat{\mathcal{C}}^n(t,y)\right|^2 dtdy\right)^{q/2}|X|^{1-q/2}\right]\\
&\le \mathcal{K}\left[\int_X |\delta(t,y)|^q dtdy + |X|^{1-q/2}\right]. \qquad (3.12)
\end{aligned}$$

By Vitali's theorem, for $n\to\infty$ we have

$$\mathcal{A}\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)\to\mathcal{A}(t,y,\mathcal{C},\nabla\mathcal{C}), \text{ strongly in } L^q\left(\mathcal{D}_T\right),$$

A comparable estimate to (3.12) can be derived for $B\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)$ when $\rho = 1+\frac{d}{d+4}$. Consequently, by applying Vitali's theorem, we obtain the following convergence:

$$B\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)\to B(t,y,\mathcal{C},\nabla\mathcal{C}), \text{ strongly in } L^\rho\left(\mathcal{D}_T\right),$$

Furthermore, it follows from the construction that the terminal condition $\mathcal{C}_T^n$ converges strongly to $\mathcal{C}_T$, in $L^2(\mathcal{D})$. The weak formulation of the PDE (3.9) with $f^n=0$, for every $\tau^*\in[0,T)$ and $\Psi\in\mathcal{C}_0^\infty\left(\mathcal{D}_T\right)$ is given as follows:

$$\begin{aligned}
&\int_{\mathcal{D}_{\tau^*}}\left[-\hat{\mathcal{C}}^n\partial_t\Psi + \left\langle\mathcal{A}\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right),\nabla\Psi\right\rangle + \left(B\left(t,y,\hat{\mathcal{C}}^n,\nabla\hat{\mathcal{C}}^n\right)-\mathcal{E}^n\right)\Psi\right](t,y)dydt\\
&\quad+\int_{\mathcal{D}}\hat{\mathcal{C}}^n\left(\tau^*,y\right)\Psi\left(\tau^*,y\right)dy - \int_{\mathcal{D}}\mathcal{C}_T^n(y)\Psi(T,y)dy = 0,
\end{aligned}$$

Applying the convergence results discussed above, we establish that $\mathcal{C}$ satisfies the weak formulation of equation (3.8):

$$\begin{aligned}
&\int_{\mathcal{D}_{\tau^*}}\left[-\mathcal{C}\partial_t\Psi + \langle\mathcal{A}(t,y,\mathcal{C},\nabla\mathcal{C}),\nabla\Psi\rangle + B(t,y,\mathcal{C},\nabla\mathcal{C})\Psi\right](t,y)dydt\\
&\quad+\int_{\mathcal{D}}\mathcal{C}\left(\tau^*,y\right)\Psi\left(\tau^*,y\right)dy - \int_{\mathcal{D}}\mathcal{C}_T(y)\Psi(T,y)dy = 0.
\end{aligned}$$

Using assumptions of uniform L^2-boundedness and the compactness of the domain, $\tilde{\mathcal{C}}^n$ weakly converges in $L^2(\mathcal{D}_T)$, while the boundary values f^n strongly converge to zero in L^2 and almost uniformly along a subsequence. Applying results from [16], $\tilde{\mathcal{C}}^n$ and $\hat{\mathcal{C}}^n$ differ negligibly as $n \to \infty$. With Vitali's theorem, $\{\tilde{\mathcal{C}}^n - \hat{\mathcal{C}}^n\}$ converges strongly to zero in $L^q(\mathcal{D}_T)$, for $q < 2$. Combined with the strong convergence of $\hat{\mathcal{C}}^n$ to $\mathcal{C}$, this proves $\tilde{\mathcal{C}}^n \to \mathcal{C}$ in $L^q(\mathcal{D}_T)$. If $\tilde{\mathcal{C}}^n$ is continuous, the Arzela-Ascoli theorem ensures uniform convergence to $\mathcal{C}$. □

3.4. Numerical Results

This part presents the results of pricing European call and put options based on (2.2). We consider the following model parameters: $r = 0.05, H = 0.9, \delta = 10^{-5}, \sigma = 0.1, \xi = 0.8, \rho = 0.7, T = 1, K = 23$. We first analyse the performance and stability of the DGM by tuning important hyper-parameters, such as the number of hidden layers, the number of hidden units per layer, and the choice of activation functions. Figure 1 shows the loss function over the epochs for networks with 2, 3, 4, and 5 hidden layers. Table 1, along with these figures, reports the average values of the loss function over the epochs, with the lowest value observed for the network with three hidden layers. While the networks with 2 and 4 hidden layers show slightly higher average losses, the network with 5 hidden layers shows an increase in the average loss, which is likely due to over-fitting caused by the excessive complexity of the network. These results indicate that the 3-layer architecture achieves the optimal balance between model complexity and training efficiency. Figure 2 examines the effect of varying the number of hidden units in a layer (for a 3-hidden layer network) with configurations of 22, 32, 42, and 52 units. The corresponding Table 2 reports the average values of the loss function and shows a decreasing trend with increasing number of hidden units. This indicates that a higher number of hidden units improves the capacity of the model to capture the complexity of the solution, although further increases may lead to computational inefficiency.

Figure 3 examines the effect of different activation functions. The corresponding Table 3 reports the average values of the loss function. The choice of activation function has a profound impact on the stability and effectiveness of the method. When using the ReLU and swish activation functions, the training process resulted in nan values for the loss function, indicating instability. Conversely, the tanh and sigmoid functions were effective in stabilizing the training process, with the tanh activation function achieving the lowest average loss. This demonstrates the importance of differentiable and bounded activation functions in ensuring numerical stability during training. The results identify 3 hidden layers and the tanh activation function as the optimal choices for the given problem. We compare the DGM results with those obtained through Monte Carlo (MC) simulations to verify their reliability. Tables 4 and 6 present the Monte Carlo results, including confidence intervals, for a selected number of simulated paths for the European call and put options, respectively. Figures 4 and 5 depict the Monte Carlo estimates for the European call and put options, respectively, using different numbers of simulated paths (from 100 to 6000). Each figure also shows the

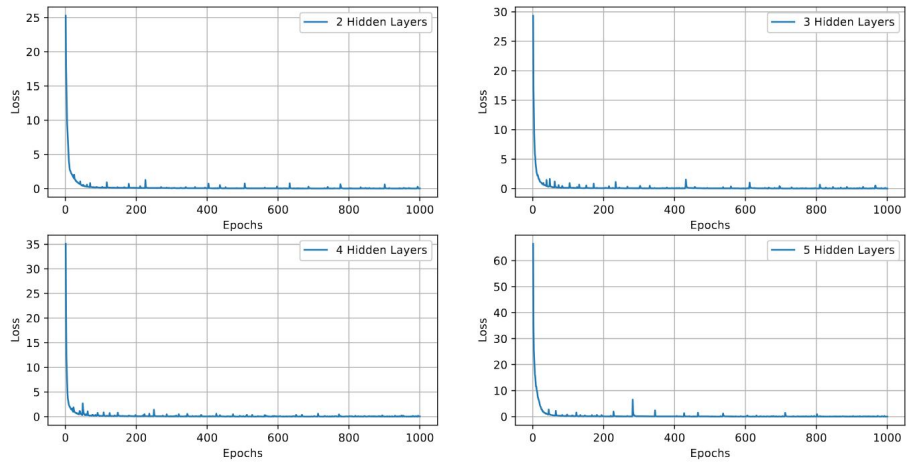

Figure 1. Loss function plotted over 1000 epochs for networks with 2, 3, 4, and 5 hidden layers.

Table 1. Average loss function values over 1000 epochs for networks with 2, 3, 4, and 5 hidden layers.

Layers	2	3	4	5
$\mathcal{J}(\tilde{\mathcal{C}})$	0.23999	0.21625	0.22483	0.48285

upper and lower bounds of the 95% confidence interval and the result obtained from the DGM. As the number of simulated paths increases, the length of the confidence intervals decreases, and the Monte Carlo estimates converge to the results of the DGM. This convergence indicates that the DGM provides a reliable solution that is consistent with the results of a widely accepted numerical technique as simulation accuracy improves.

Tables 5 and 7 provide a detailed comparison of option prices calculated using the DGM and Monte Carlo simulation for European call and put options, respectively. We present the results for the underlying asset prices $S = 20, 21, \cdots, 26$, and volatilities $V = 0.01, 0.05, 0.1, 0.2$. The tables also include the relative differences between the two methods, as shown by the Δ. The small relative differences observed in all cases emphasize the reliability and accuracy of the DGM when approximating option prices. The results reveal several important trends. For both call and put options, the option price increases with increasing volatility, which is consistent with expectations given the dynamics of the model. For call options, the option price increases with increasing underlying asset price, consistent with the positive correlation between the asset value and the intrinsic value of the call option. Conversely, as S increases, put option prices decrease, reflecting the decreasing value of the put option with a higher strike price.

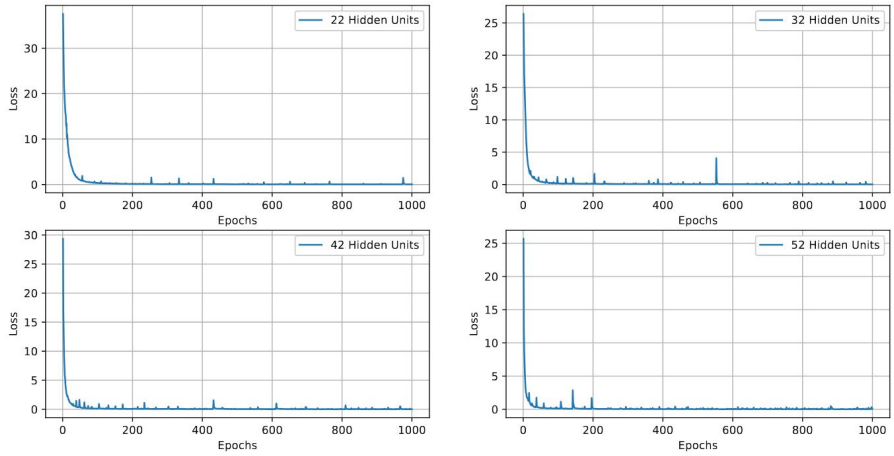

Figure 2. Loss function plotted over 1000 epochs for networks with varying hidden units in a single layer (3-hidden layer network).

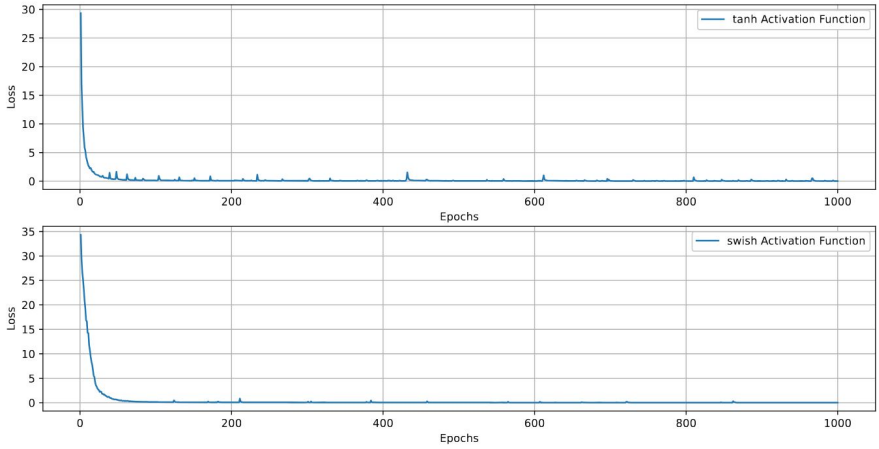

Figure 3. Loss function plotted over 1000 epochs for networks with different activation functions (3-hidden layer network with 42 units in a layer).

Table 2. Average loss function values over 1000 epochs for networks with varying hidden units in a single layer (3-hidden layer network).

Units	22	32	42	52
$\mathcal{J}(\tilde{\mathcal{C}})$	0.47853	0.27873	0.21625	0.17041

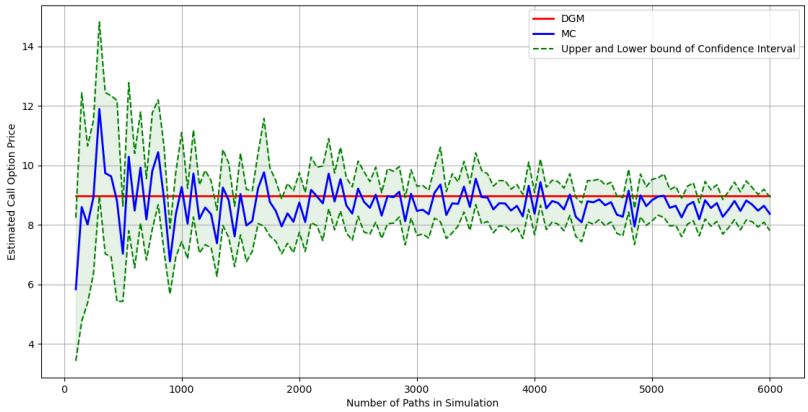

Figure 4. Comparison of the European call option prices obtained using the Monte Carlo method with varying numbers of simulated paths and the DGM.

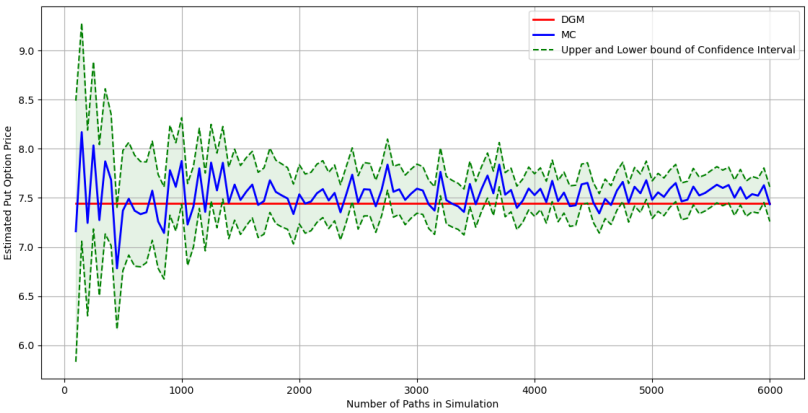

Figure 5. Comparison of the European put option prices obtained using the Monte Carlo method with varying numbers of simulated paths and the DGM.

Table 3. Average loss function values over 1000 epochs for different activation functions.

Activation function	tanh	sigmoid	ReLU	swish
$\mathcal{J}(\tilde{\mathcal{C}})$	0.21625	0.41287	nan	nan

Table 4. Monte Carlo simulation results for European call option prices with varying numbers of simulated paths.

Number of simulations	Monte-Carlo estimation	Confidence interval
100	5.83285	[3.42766, 8.23805]
1000	9.27054	[7.43682, 11.1043]
2000	8.74916	[7.74812, 9.75020]
3000	8.46939	[7.64031, 9.29847]
4000	8.37327	[7.66292, 9.08363]
5000	8.83407	[8.13619, 9.53195]
6000	8.37080	[7.80574, 8.93586]

Table 5. Comparison of European call option prices obtained using the deep Galerkin and Monte Carlo methods.

$S \setminus V$	0.2	0.1	0.05	0.01
20	DGM = 7.76522 MC = 7.88206 $\Delta = 0.01482$	DGM = 7.62059 MC = 7.18091 $\Delta = 0.06123$	DGM = 7.43625 MC = 7.04054 $\Delta = 0.05620$	DGM = 7.05548 MC = 6.76128 $\Delta = 0.04351$
21	DGM = 8.28679 MC = 8.13805 $\Delta = 0.01828$	DGM = 8.14801 MC = 7.93189 $\Delta = 0.02725$	DGM = 7.97076 MC = 7.48382 $\Delta = 0.06506$	DGM = 7.60339 MC = 7.22176 $\Delta = 0.05284$
22	DGM = 8.80918 MC = 8.44704 $\Delta = 0.04287$	DGM = 8.67664 MC = 8.41046 $\Delta = 0.03165$	DGM = 8.50705 MC = 8.33141 $\Delta = 0.02108$	DGM = 8.15446 MC = 7.97480 $\Delta = 0.02253$
23	DGM = 9.33228 MC = 9.38891 $\Delta = 0.00603$	DGM = 9.20626 MC = 8.97159 $\Delta = 0.02616$	DGM = 9.04475 MC = 8.74855 $\Delta = 0.03386$	DGM = 8.70803 MC = 8.52758 $\Delta = 0.02116$
24	DGM = 9.85619 MC = 10.05744 $\Delta = 0.02001$	DGM = 9.73685 MC = 9.90044 $\Delta = 0.01652$	DGM = 9.58370 MC = 9.87053 $\Delta = 0.02906$	DGM = 9.26367 MC = 9.59258 $\Delta = 0.03429$
25	DGM = 10.38101 MC = 10.56740 $\Delta = 0.01764$	DGM = 10.26847 MC = 10.44642 $\Delta = 0.01703$	DGM = 10.123874 MC = 10.39666 $\Delta = 0.02624$	DGM = 9.82111 MC = 10.09279 $\Delta = 0.02692$
26	DGM = 10.90678 MC = 11.47258 $\Delta = 0.04932$	DGM = 10.80108 MC = 11.20562 $\Delta = 0.03610$	DGM = 10.66517 MC = 11.08372 $\Delta = 0.03776$	DGM = 10.38011 MC = 10.48856 $\Delta = 0.01034$

Table 6. Monte Carlo simulation results for European put option prices with varying numbers of simulated paths.

Number of simulations	Monte-Carlo estimation	Confidence interval
100	7.16067	[5.83140, 8.48994]
1000	7.87572	[7.43559, 8.31586]
2000	7.53658	[7.23494, 7.83822]
3000	7.59413	[7.34420, 7.84406]
4000	7.52922	[7.31249, 7.74596]
5000	7.48193	[7.29040, 7.67347]
6000	7.43637	[7.25980, 7.61294]

Table 7. Comparison of European put option prices obtained using the deep Galerkin and Monte Carlo methods.

$S \setminus V$	0.2	0.1	0.05	0.01
20	DGM = 9.35534 MC = 9.36016 $\Delta = 0.00051$	DGM = 9.23118 MC = 8.94775 $\Delta = 0.03168$	DGM = 9.06890 MC = 8.82345 $\Delta = 0.02782$	DGM = 8.72064 MC = 8.49938 $\Delta = 0.02603$
21	DGM = 8.91192 MC = 8.94330 $\Delta = 0.00351$	DGM = 8.79064 MC = 8.68449 $\Delta = 0.01222$	DGM = 8.63188 MC = 8.67936 $\Delta = 0.00547$	DGM = 8.29017 MC = 8.17124 $\Delta = 0.01455$
22	DGM = 8.47241 MC = 8.80256 $\Delta = 0.03751$	DGM = 8.35423 MC = 8.51050 $\Delta = 0.01836$	DGM = 8.19938 MC = 8.19100 $\Delta = 0.00102$	DGM = 7.86546 MC = 8.00124 $\Delta = 0.01697$
23	DGM = 8.03658 MC = 8.34524 $\Delta = 0.03699$	DGM = 7.92171 MC = 8.11800 $\Delta = 0.02418$	DGM = 7.77112 MC = 7.81782 $\Delta = 0.00597$	DGM = 7.44609 MC = 7.53224 $\Delta = 0.01144$
24	DGM = 7.60443 MC = 8.28617 $\Delta = 0.08227$	DGM = 7.49304 MC = 7.84670 $\Delta = 0.04507$	DGM = 7.34702 MC = 7.78723 $\Delta = 0.05653$	DGM = 7.03182 MC = 7.33806 $\Delta = 0.04173$
25	DGM = 7.17588 MC = 7.83071 $\Delta = 0.08362$	DGM = 7.06807 MC = 7.66128 $\Delta = 0.07743$	DGM = 6.92683 MC = 7.46775 $\Delta = 0.07243$	DGM = 6.62218 MC = 6.92774 $\Delta = 0.04411$
26	DGM = 6.75072 MC = 7.55076 $\Delta = 0.10595$	DGM = 6.64653 MC = 7.46866 $\Delta = 0.11008$	DGM = 6.51019 MC = 7.16199 $\Delta = 0.091001$	DGM = 6.21661 MC = 6.67314 $\Delta = 0.06841$

4. Portfolio optimization with Machine Learning

In this section, we discuss methods for determining the optimal portfolio that includes options. The integration of the long memory SLV model with machine learning techniques forms a key contribution of our portfolio optimization framework. In general, the option PDE solved numerically using the DGM, serves two essential purposes. First, it provides more accurate prices for European call and put options under realistic market assumptions, including long-range dependence, leverage effects, and stochastic-local volatility dynamics. These option prices directly influence the expected return term $\mu_i + (R_{call})_i$ and the transaction cost components in the CCMV objective function (Equation 4.2). Second, the SLV-DGM framework generates improved estimates of volatility surfaces and risk measures, which are then used as input features for the machine learning classifiers. Specifically, we train Random Forest, KNN, and SVM models using historical daily price data from 2017 to 2022, where the feature set includes not only standard financial indicators (returns, variances) but also SLV-derived implied volatilities and option-implied risk metrics. Each machine learning method then classifies stocks into risk categories (low risk, high return, or neither) for the out-of-sample year 2023. These classifications serve as filters that constrain the investment universe before applying the CCMV model. For risk-averse investors, we retain only stocks classified as low risk by the machine learning models; for risk-seeking investors, we retain only those classified as high return. The final optimal portfolios are thus the result of a two-stage integration: the SLV model en-

riches the financial inputs with realistic option pricing and volatility dynamics, while the machine learning classifiers provide data-driven, out-of-sample risk-return predictions. This hybrid approach ensures that the portfolio optimization is simultaneously grounded in rigorous financial theory and adaptive to empirical market patterns.
In recent years, significant attention has been directed toward the use of machine learning techniques, such as SVM and deep learning models, for stock market prediction which highlighted the promise and performance of these methods in handling complex financial datasets. This study aims to present a comprehensive comparative analysis of SVM and deep learning models using a dataset comprising stocks spanning a six-year period, sourced from Yahoo Finance. The subsequent sections provide a brief overview of these predictive models, setting the stage for a detailed examination of their capabilities and limitations in stock market forecasting.

4.1. Support Vector Machine

Support Vector Machine [6] is a powerful supervised learning model that finds the optimal separating hyperplane to maximize the margin between different classes. In portfolio optimization, SVM is particularly adept at binary classification tasks that are central to risk management and asset screening. Common applications include distinguishing between "high return" vs. "low return" assets or "risky" vs. "safe" assets. By using kernel tricks (e.g., radial basis function or polynomial kernels), SVM can capture non-linear decision boundaries that simpler linear models miss, which is crucial given the often non-linear relationships in financial time series. The margin maximization principle of SVM also confers good generalization performance, meaning that an SVM trained on historical data is less likely to overfit to noise and more likely to perform reliably on unseen future market conditions. For portfolio optimization, SVM can be used as a pre-processing filter: only assets that the SVM confidently classifies as belonging to the desired risk/return category are considered for inclusion. This approach reduces the investment universe to a high-conviction set, thereby improving the efficiency of subsequent optimization steps.

Consider a binary classification problem defined by the dataset $D=\{(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)\}$, where each row vector $\boldsymbol{y}_i$ represents a sample, and $\boldsymbol{y}_i \in \{-1, 1\}$ for $i = 1, \ldots, l$. The objective of the SVM is to determine an optimal hyperplane, represented as $w^T x + b$, that effectively separates the training samples based on their labels. This hyperplane can subsequently be employed to classify new, unseen data points. The soft-margin SVM formulation is mathematically expressed as follows [6]:

$$\begin{aligned} \min_{w,b,\eta} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{l}\eta_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \eta_i, \quad i = 1, \ldots, l, \\ & \eta_i \geq 0. \end{aligned}$$

where positive constant C acts as a trade-off hyperparameter, balancing margin maximization and misclassification tolerance. The margin is defined by two parallel hyperplanes: $w^T\boldsymbol{x} + b = 1$ and $w^T\boldsymbol{x} + b = -1$. Positive samples ($y = 1$) must satisfy $w^T\boldsymbol{x} + b \geq 1$, and negative samples ($y = -1$) must satisfy $w^T\boldsymbol{x} + b \leq -1$. For a new point $\boldsymbol{x}$, the predicted label is given by the sign of $w^T\boldsymbol{x} + b$.
In practice, real-world data distributions often make it difficult to separate data points using linear SVMs and their extensions. To address this, a nonlinear mapping function $\phi(\boldsymbol{x})$ is applied, which projects the original data points $\boldsymbol{x}$ into a higher-dimensional feature space $\mathcal{H}$, where linear separation becomes more feasible. Mercer's theorem ensures the existence of a **kernel function** $K(\boldsymbol{x}, \boldsymbol{x}')$, which computes the inner product $\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$ in $\mathcal{H}$ without explicitly performing the transformation. A commonly used example is the **Gaussian kernel** (or Radial Basis Function kernel), defined as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|^2\right),$$

where $\gamma > 0$ is a parameter that controls the kernel's spread.
where $\sigma > 0$ is a hyperparameter that controls the influence of the similarity between data points. The dual formulation of SVM using the kernel function is as follows [6]:

$$\begin{aligned} \max_{\alpha,\,\beta} \quad & -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j K(x_i, x_j) + \sum_{j=1}^{l} \alpha_j, \\ \text{s.t.} \quad & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & 0 \leq \alpha_i \leq \mathcal{C}, \quad i = 1, \ldots, l. \end{aligned} \tag{4.1}$$

Let $\alpha^* = (\alpha_1^*, \ldots, \alpha_l^*)^T$ be a solution to the problem (4.1). If there exists a component of α^* such that $\alpha_j^* \in (0, C)$, then (w^*, b^*) of the primal problem (4.1) can be obtained via following way:

$$\begin{aligned} w^* &= \sum_{i=1}^{l} \alpha_i^* y_i x_i, \\ b^* &= y_j - \sum_{i=1}^{l} y_i \alpha_i^* (x_i^T x_j). \end{aligned}$$

The label of the new instance x is determined via following decision function:

$$f(x) = sign_{k=1,2}(w^T x + b).$$

The Radial Basis Function (RBF) kernel, defined as $K(x, x') = e^{\|x-x'\|^2/2\sigma^2}$ is employed, with the hyperparameter σ selected from the set $\{2^i | i = -8, \ldots, 12\}$. Additionally, the hyperparameter $\mathcal{C}$ is chosen from the set $\{2^i |, i = -12, \ldots, 12\}$.

4.2. Random Forest

Random Forest [5] is a machine learning method that builds many decision trees using random samples of data and random subsets of features. Each tree gives a prediction, and the final output is the majority vote (for classification) or the average (for regression). In portfolio optimization, this method helps investors in two main ways. First, it can rank financial indicators such as moving averages, trading volume, or past volatility based on how useful they are for predicting future price movements. This helps investors focus on the most important signals and ignore irrelevant noise. Second, because Random Forest averages the results of many trees, it is much less likely to learn false patterns from random fluctuations in historical data. This makes the model more reliable when applied to new, unseen market conditions. Investors can use Random Forest to sort assets into simple categories such as low risk, medium risk, or high return. They can then choose to invest only in assets that consistently appear in the desired category across most of the decision trees. This approach leads to more stable portfolios that perform better in real-world situations.

The method improves classification accuracy by combining many decision trees and reducing over-fitting through two sources of randomness: bootstrapped data subsets and random feature selection at each split, a technique known as feature bagging. Predictions are made by majority voting, which makes the model less sensitive to outliers and noise. Random Forest also provides a measure of feature importance, which is useful for interpretability. In our experiments, we set the number of trees (n_estimators) to 100. This number is large enough to average out prediction errors and lower variance while keeping the model stable. Each tree is trained on a different bootstrapped sample of the training data. To make our results reproducible, we fix the random seed (random_state) to 2. This controls all random processes including data sampling and feature selection during tree construction. By doing so, we ensure that running the same code on the same data always produces identical results, which is essential for academic transparency and fair comparison with other methods.

4.3. K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a simple learning method [2] that does not make any assumptions about the shape or form of the underlying data. When a new asset needs to be classified, KNN looks at the k most similar assets from the historical record. Similarity is usually measured by a distance metric such as Euclidean distance. In portfolio optimization, this approach is valuable because it is easy to understand and explain to investors. KNN does not assume a linear relationship between past data and future performance, so it can handle complex and changing market behaviours. Practically, an investor can use KNN to find assets that historically behaved like the current candidate in terms of volatility, trading volume, or price momentum. The classification of the new asset is then decided by the majority label among those k neighbors. This method also helps build diversified portfolios because if two assets are very close in the feature space, they are likely similar, and including both may

not add much diversification. By checking distances to existing holdings, KNN can suggest different assets. The parameter k controls a trade-off: a small k captures very local patterns and is useful for short-term decisions, while a large k gives more stable and reliable classifications suitable for long-term investment strategies.

In the paper, we use the K-Nearest Neighbors algorithm as a simple but effective tool. It is especially well-suited for binary classification problems like ours, where each stock must be labelled as belonging to a certain risk type or not. The algorithm works entirely based on proximity in the feature space. For a new, unlabelled data point, KNN finds the k closest points from the training set using a distance metric, which we take to be Euclidean distance. The new point is then assigned the label that appears most frequently among those k neighbors. By focusing only on local neighbors, KNN naturally captures patterns and trends in risk behavior over multiple years. This property makes the method highly adaptable to the structure of our dataset, where temporal features such as price changes over time help predict the likelihood of a specific risk category in the following year. No complex model fitting or parameter estimation is required, which keeps the method transparent and computationally efficient.

4.4. Evaluation

In this section, we evaluate the classification performance of machine learning models, namely SVM, RF, KNN in aspect of classification accuracy. To optimize the regression models, we fine-tuned the hyperparameters using the input training dataset, where each instance corresponds to an index, and its associated features represent the daily price data from 2017 to 2021 for that index. To evaluate the compared models accuracy, we used 10 fold cross validation. In Tables 8 and 9, we present the confusion matrices. A confusion matrix provides a detailed breakdown of the model's predictions, highlighting both correctly classified samples and misclassifications. For a 3-class classification problem, the confusion matrix is typically structured as a 3×3 table, where the **rows** represent the true classes, and the **columns** represent the predicted classes. However, for the SVM model, the confusion matrix is presented as a 2×2 table due to the specific approach applied for the multi-label classification task. Although Support Vector Machines (SVMs) have been extended to handle multi-label tasks through strategies such as one-vs-one, one-vs-all, and one-vs-one-vs-rest, we adopted a different methodology tailored to the nature of our dataset. Given that our portfolio optimization problem involves two types of risks, we utilized the SVM model to classify these risks in two separate binary classification tasks. The first task distinguishes between *low* and *non-low* risks (labels 1 and 0), while the second task differentiates between *high* and *non-high* risks (labels 1 and 0).

Table 8. **Confusion Matrices for K-Nearest Neighbour and Random Forest**

Model	Actual	Low risk	None of them	High return
K-Nearest Neighbour	**Low risk**	4	6	0
	None of them	9	50	0
	High return	1	9	0
Random Forest	**Low risk**	4	6	0
	None of them	13	45	1
	High return	2	8	0

Table 9. **Confusion Matrices for Low Risk and High return**

Risk Level	Actual	Predicted 1	Predicted 0
Low Risk	**Actual 1**	3	16
	Actual 0	7	53
High Return	**Actual 1**	2	9
	Actual 0	8	60

Table 10. **Optimal stock portfolio using machine learning methods.**

	K-NN L-R	K-NN H-R	R-F L-R	R-F H-R	SVM L-R	SVM H-R	Real Data L-R	Real Data H-R		K-NN L-R	K-NN H-R	R-F L-R	R-F H-R	SVM L-R	SVM H-R	Real Data L-R	Real Data H-R
PANW	0	0	0	0	0	1	0	1	CHTR	0	0	0	0	0	0	0	0
PAYX	0	0	0	0	1	0	1	0	CDNS	0	0	0	0	0	1	0	1
MDLZ	1	0	1	0	1	0	1	0	ADBE	0	0	0	0	0	1	0	1
ODFL	0	0	0	0	0	0	0	0	BIIB	0	0	0	0	0	0	0	0
MCHP	0	0	0	0	0	0	0	0	AVGO	0	0	0	0	0	0	0	1
ORLY	0	0	0	0	0	1	0	0	BKR	0	0	0	0	1	0	0	0
NX	0	0	0	0	0	0	0	0	STX	0	0	1	0	1	0	0	0
IDXX	0	0	0	0	0	1	0	0	PTC	0	0	0	0	0	0	0	0
KLAC	0	0	0	0	0	1	0	0	TER	0	0	0	0	0	0	0	0
KDP	0	0	0	0	0	0	1	0	CHKP	0	0	0	0	0	0	0	0
MAR	0	0	0	0	0	0	0	0	HBAN	0	0	0	0	1	0	0	0
MRVL	0	0	0	0	0	0	0	0	COO	0	0	1	0	0	0	0	0
MELI	0	0	0	0	0	1	0	1	PFG	0	0	0	0	0	0	0	0
KHC	0	0	0	0	1	0	0	0	STLD	0	0	0	0	0	0	0	0
LULU	0	0	0	0	0	1	0	0	EXPE	0	0	0	0	1	0	0	0
LRCX	0	0	0	1	0	1	0	1	UAL	0	0	1	0	0	0	0	0
FAST	0	0	0	0	1	0	0	0	WBD	0	0	0	0	0	0	0	0
ISRG	1	0	1	0	0	0	0	0	ZBRA	0	0	0	0	0	1	0	0
TTD	0	0	1	0	0	0	0	1	HOLX	0	0	0	0	0	0	0	0
INTU	0	0	0	0	0	1	0	0	VRSN	1	0	1	0	1	0	1	0
FTNT	0	0	0	0	1	0	0	0	NTRS	0	0	0	0	0	0	0	0
INTC	1	0	1	0	1	1	0	1	LPLA	0	0	1	0	0	0	0	0
HON	1	0	1	0	1	0	1	0	SSNC	0	0	0	0	0	0	0	0
ILMN	0	0	0	0	0	0	0	0	ALGN	0	0	0	0	0	1	0	0
GILD	0	0	1	0	0	0	1	0	ENTG	0	0	0	0	0	0	0	0
META	1	0	1	0	0	0	0	1	GEN	0	0	0	0	1	0	0	0
CMCSA	1	0	1	0	1	0	0	0	JBHT	0	0	0	0	0	0	0	0
CTAS	0	0	0	0	0	1	0	0	NTNX	0	0	1	0	0	1	0	1
CSCO	1	0	1	0	1	0	0	0	NTRA	0	0	0	0	0	0	0	0
FANG	0	0	0	0	0	0	0	0	UTHR	0	0	0	0	1	0	1	0
EXC	1	0	0	0	1	0	1	0	LNT	1	0	1	0	0	0	1	0
DLTR	0	0	0	0	0	1	0	0	AKAM	0	0	0	0	1	0	0	0
CSX	1	0	0	0	1	0	1	0	DLTR	0	0	0	0	0	1	0	0
CTSH	0	0	1	0	0	0	0	0	TRMB	0	0	0	0	0	0	0	0
DXCM	0	0	0	0	0	1	0	0	NWS	0	0	0	0	1	0	0	0
AZN	1	0	0	0	1	0	0	0	NDSN	1	0	1	0	1	0	0	0
LIN	0	0	0	0	1	0	0	0	ENPH	0	0	0	0	1	1	0	0
CSGP	0	0	0	0	0	0	0	0	CPB	0	0	0	0	0	0	0	0
BKNG	0	0	0	0	0	0	0	0	EVRG	0	0	1	0	0	0	0	0
CPRT	1	0	0	0	1	0	0	0									

Comparing Tables 8 and 9, it can be seen that SVM model predicted more low risk and high return instances than other models. we set hyperparameter of SVM based on these results and predict the type of risks for year 2023 where our input includes indexes and its features is daily prices from 2018 till 2022 and the results have shown in the Table 10. According to Table 10, the K-Nearest Neighbors (KNN) and Random Forest (RF) models correctly identified 6 and 4 out of 10 true low-risk sets, respectively, while the Support Vector Machine (SVM) achieved a higher performance with 7 out of 10 correctly classified low-risk sets. In the case of high-risk sets, KNN failed to distinguish any high-risk sets from non-high-risk ones, and RF accurately identified only 1 out of 10 true high-risk sets. In contrast, SVM demonstrated superior performance, correctly predicting 7 out of 10 true high-risk sets.

4.5. Portfolio optimization under the neural network methods and optimization models

Here, we introduce a strategy aimed at enhancing the efficiency of machine learning models through the use of the CCMV model. At first, we use this model to derive the optimal stock portfolio for the years 2017 to 2022 from the Russell index according to the risk tolerance of investors. Next, we identify the stocks that are in the optimal portfolio over this six-year period. Finally, by integrating these findings with the results of machine learning techniques, we create the optimal stock portfolio for 2023 and compare it with the optimal portfolio obtained considering real market data. The general form of the CCMV model is as follows:

$$\begin{aligned}
\min \quad & \lambda\Big(\sum_{i=1}^{N}\sum_{j=1}^{N} x_i x_j \sigma_{i,j}\Big) - (1-\lambda)\Big(\sum_{i=1}^{N} x_i\,(\mu_i + (R_{call})_i) - \sum_{i=1}^{N} O_i|x_i| - \sum_{i=1}^{N} k_i|x_i - x_i^0|\Big) \\
s.t. \quad & \sum_{j=1}^{N} x_i = 1, \qquad (4.2)\\
& \sum_{i=1}^{N} z_i = K, \\
& \varepsilon_i z_i \le x_i \le \delta_i z_i, \qquad i = 1, ..., N, \\
& \mu_i x_i \ge 0, \qquad i = 1, ..., N, \\
& z_i \in \{0, 1\}, \qquad i = 1, ..., N.
\end{aligned}$$

where x_i denotes the proportion of total funds invested in the ith stock for $i = 1, \ldots, N$ and N represents the number of available stocks. The expected return of the ith stock is indicated by μ_i, while σ_i represents its standard deviation. The cardinality constraint, which specifies the maximum number of assets allowed in the portfolio, is denoted by K. Additionally, ε_i and δ_i correspond to the lower and upper bounds of the ith stock, respectively. The covariance between stocks i and j is expressed as $\sigma_{i,j}$, and the risk aversion parameter, which takes values within the interval $\lambda \in [0, 1]$, is also included in this model.

In the below table, we created a portfolios based on the risk and return of stocks and

related options. In Table 11, we use the CCMV model 4.2 and obtained the optimal portfolio contains 10 stocks with their options for risk-averse investors ($\lambda = 1$) for the years 2017 to 2022. After that, we categorized the stocks that were in the optimal stock portfolio in these years as follows. We have classified stocks that were included in the optimal portfolio only twice with code 11 and stocks that were included in the optimal portfolio more than twice with code 1.

Table 11. Optimal stock portfolio for risk-averse investors for 2017 to 2022 using the CCMV model.

	2017	2018	2019	2020	2021	2022	Selected stocks		2017	2018	2019	2020	2021	2022	Selected stocks
PANW	0	0	0	0	0	0	0	CHTR	0	0	0	0	0	0	0
PAYX	0	0	1	0	0	1	11	CDNS	0	0	0	0	0	0	0
MDLZ	1	1	0	1	1	1	1	ADBE	0	0	0	0	0	0	0
ODFL	0	0	0	0	0	1	0	BIIB	0	0	0	1	0	0	0
MCHP	0	0	0	0	0	0	0	AVGO	0	1	0	0	0	0	0
ORLY	0	0	0	0	0	0	0	BKR	0	0	0	0	0	0	0
NXPI	1	0	0	0	0	0	0	STX	0	0	0	0	0	0	0
IDXX	0	0	0	0	0	0	0	PTC	0	0	0	0	0	1	0
KLAC	0	0	0	0	0	0	0	TER	0	0	0	0	0	0	0
KDP	1	0	1	1	1	0	1	CHKP	0	0	1	0	1	0	11
MAR	0	0	0	0	0	0	0	HBAN	1	0	1	0	1	0	1
MRVL	0	0	0	0	0	0	0	COO	0	0	0	1	1	0	11
MELI	0	0	0	0	0	0	0	PFG	1	0	1	0	0	0	11
KHC	0	0	0	0	0	0	0	STLD	0	0	0	0	0	0	0
LULU	0	0	0	0	0	0	0	EXPE	0	0	0	0	0	0	0
LRCX	0	0	0	0	0	0	0	UAL	0	0	1	0	0	0	0
FAST	0	0	0	0	0	0	0	WBD	0	0	0	0	0	0	0
ISRG	0	0	0	0	0	0	0	ZBRA	0	0	0	0	0	0	0
TTD	0	0	0	0	0	0	0	HOLX	0	1	1	0	0	1	1
INTU	0	0	0	0	0	0	0	VRSN	0	0	0	1	0	0	0
FTNT	0	0	0	0	0	0	0	NTRS	1	0	0	1	0	0	11
INTC	0	0	0	0	0	0	0	LPLA	0	1	0	0	0	0	0
HON	0	1	0	0	1	0	11	SSNC	0	0	0	0	0	0	0
ILMN	0	0	0	0	0	0	0	ALGN	0	0	0	0	0	0	0
GILD	0	0	1	0	1	0	11	ENTG	0	0	0	0	0	0	0
META	0	0	0	0	0	0	0	GEN	0	0	0	1	0	0	0
CMCSA	1	0	0	0	0	0	0	JBHT	0	0	0	0	0	1	0
CTAS	0	0	0	0	0	0	0	NTNX	0	0	0	0	0	0	0
CSCO	0	0	0	0	0	0	0	NTRA	0	0	0	0	0	0	0
FANG	0	0	0	0	0	0	0	UTHR	0	0	0	0	0	0	0
EXC	1	0	1	0	0	0	0	LNT	1	1	0	1	0	1	1
DLTR	0	0	0	0	0	0	0	AKAM	0	0	0	0	1	0	0
CSX	0	0	0	0	0	0	0	DLTR	0	0	0	0	0	0	0
CTSH	0	0	0	0	0	0	0	TRMB	0	0	0	0	0	0	0
DXCM	0	0	0	0	0	0	0	NWS	0	0	0	0	0	0	0
AZN	0	1	0	1	0	0	11	NDSN	0	1	0	0	0	1	0
LIN	0	1	0	0	0	0	0	ENPH	0	0	0	0	0	0	0
CSGP	0	0	0	0	1	0	0	CPB	0	0	0	1	0	0	0
BKNG	0	0	1	0	1	0	11	EVRG	1	1	0	0	0	1	1
CPRT	0	0	0	0	0	1	0								

In Table 12, we considered the stock prices for each year and calculated their price variance, which is a measure of volatility or risk. Then we sorted the stocks from small to large according to volatility and considered the first third as low volatility stocks and labelled them with code 1. We also labelled the rest of the shares with code zero. Finally, we identified stocks that were selected as low volatility stocks three or more times as code 1 and considered them as selected stocks.

Table 12. **Categorizing stocks based on low volatility.**

	2017	2018	2019	2020	2021	2022	Selected stocks		2017	2018	2019	2020	2021	2022	Selected stocks
PANW	0	0	0	0	0	0	0	CHTR	0	0	0	1	1	0	0
PAYX	1	1	1	1	1	1	1	CDNS	1	1	0	1	0	0	1
MDLZ	1	1	1	1	1	1	1	ADBE	0	0	0	1	0	0	0
ODFL	1	0	0	1	0	1	1	BIIB	0	0	0	1	0	0	0
MCHP	0	0	0	0	1	0	0	AVGO	0	1	0	0	0	0	0
ORLY	0	1	1	0	1	1	1	BKR	0	0	0	0	0	0	0
NXPI	1	0	0	0	1	0	0	STX	0	0	0	1	0	0	0
IDXX	0	0	1	1	0	0	0	PTC	0	0	0	0	0	1	0
KLAC	0	0	0	0	1	0	0	TER	0	0	0	0	0	0	0
KDP	1	0	1	1	1	1	1	CHKP	0	1	1	1	1	1	1
MAR	1	1	0	0	0	0	0	HBAN	1	0	1	0	1	1	1
MRVL	0	0	0	0	0	0	0	COO	0	1	1	1	1	1	1
MELI	0	0	0	0	0	0	0	PFG	1	0	1	0	1	0	1
KHC	1	0	0	0	0	1	0	STLD	0	0	0	0	0	0	0
LULU	0	0	0	0	0	0	0	EXPE	0	0	0	0	0	0	0
LRCX	0	0	0	0	0	0	0	UAL	0	0	1	0	0	0	0
FAST	0	0	0	1	1	1	1	WBD	0	0	1	0	0	0	0
ISRG	1	0	0	1	0	0	0	ZBRA	0	0	0	0	0	0	0
TTD	0	0	0	0	0	0	0	HOLX	1	1	1	0	0	1	1
INTU	0	1	0	0	1	0	0	VRSN	1	1	0	1	1	0	1
FTNT	0	0	0	0	0	0	0	NTRS	1	1	0	1	0	1	1
INTC	0	1	0	0	0	0	0	LPLA	0	1	0	0	0	1	0
HON	1	1	1	1	0	1	1	SSNC	1	0	0	0	0	1	0
ILMN	0	0	0	0	0	0	0	ALGN	0	0	0	0	0	0	0
GILD	0	0	1	1	1	0	1	ENTG	0	0	0	0	1	0	0
META	1	0	0	0	0	0	0	GEN	0	0	0	1	0	1	0
CMCSA	1	1	1	0	1	1	1	JBHT	1	1	0	1	0	1	1
CTAS	1	0	0	0	1	0	0	NTNX	0	0	0	0	0	0	0
CSCO	1	0	0	0	1	1	1	NTRA	0	0	0	0	0	0	0
FANG	0	0	0	0	0	0	0	UTHR	0	0	0	0	0	0	0
EXC	1	1	1	1	1	1	1	LNT	1	1	1	1	0	1	1
DLTR	0	1	0	0	0	0	0	AKAM	0	0	1	1	1	1	1
CSX	0	0	1	0	0	0	0	DLTR	0	1	0	0	0	0	0
CTSH	0	1	1	0	0	1	1	TRMB	1	0	0	0	1	0	0
DXCM	0	0	0	0	0	0	0	NWS	1	1	0	0	0	0	0
AZN	0	1	1	1	1	1	1	NDSN	0	1	0	0	1	1	1
LIN	1	1	1	1	1	1	1	ENPH	0	0	0	0	0	0	0
CSGP	0	0	0	0	1	1	0	CPB	0	0	1	1	1	1	1
BKNG	0	1	1	0	1	0	1	EVRG	1	1	1	0	1	1	1
CPRT	1	0	1	0	0	1	1								

In Table 13, we use the CCMV model 4.2 and obtained the optimal portfolio contains 10 stocks with their options for risky investors ($\lambda = 0$) for the years 2017 to 2022. After that, we categorized the stocks that were in the optimal stock portfolio in these years as follows. We classified stocks that were included in the optimal portfolio only once with code 11 and stocks that were included in the optimal portfolio more than once with code 1.

In Table 14, we considered the stock prices for each year and calculated their return. Then we sorted the stocks from large to small according to return and considered the first third as high return stocks and labelled them with code 1. We also labelled the rest of the shares with code zero. We classified stocks that were included in the optimal portfolio only twice with code 11 and stocks that were included in the optimal portfolio more than twice with code 1.
In Table 15, we considered the stock prices for each year and calculated their price variance. Then. we sorted the stocks from large to small according to volatility and considered the first third as the high volatility stocks and labelled them with code 1. We also labelled the rest of the shares with code zero. Finally, we classified stocks

Table 13. Optimal stock portfolio for risk-averse investors for 2017 to 2022 using the CCMV model.

	2017	2018	2019	2020	2021	2022	Selected stocks		2017	2018	2019	2020	2021	2022	Selected stocks
PANW	0	0	0	0	1	0	11	CHTR	0	0	0	0	0	0	0
PAYX	0	0	0	0	0	0	0	CDNS	0	0	0	1	0	0	11
MDLZ	0	0	0	0	0	0	0	ADBE	1	0	0	0	0	0	11
ODFL	0	0	0	0	1	0	11	BIIB	0	0	0	0	0	1	11
MCHP	0	0	0	0	0	0	0	AVGO	0	0	0	0	1	0	11
ORLY	0	1	0	0	0	1	1	BKR	0	0	0	0	0	0	0
NXPI	0	0	0	0	0	0	0	STX	0	0	0	0	1	0	11
IDXX	0	0	0	1	0	0	11	PTC	0	1	0	0	0	0	11
KLAC	0	0	1	0	1	0	1	TER	1	0	1	0	0	0	1
KDP	0	0	0	0	0	0	0	CHKP	0	0	0	0	0	0	0
MAR	0	0	0	0	0	0	0	HBAN	0	0	0	0	0	0	0
MRVL	0	0	0	1	1	0	1	COO	0	0	0	0	0	0	0
MELI	1	0	1	1	0	0	1	PFG	0	0	0	0	0	1	11
KHC	0	0	0	0	0	0	0	STLD	0	0	0	0	1	1	1
LULU	0	1	0	0	0	0	11	EXPE	0	0	0	0	0	0	0
LRCX	1	0	1	0	0	0	1	UAL	0	0	0	0	0	0	0
FAST	0	0	0	0	0	0	0	WBD	0	0	0	0	0	0	0
ISRG	1	0	0	0	0	0	11	ZBRA	0	1	0	0	0	0	11
TTD	1	1	1	1	0	0	1	HOLX	0	0	0	0	0	0	0
INTU	0	0	0	0	1	0	11	VRSN	0	1	0	0	0	0	11
FTNT	0	1	0	0	1	0	1	NTRS	0	0	0	0	0	0	0
INTC	0	0	0	0	0	0	0	LPLA	0	0	0	0	0	1	11
HON	0	0	0	0	0	0	0	SSNC	0	0	0	0	0	0	0
ILMN	1	0	0	0	0	0	11	ALGN	1	0	0	1	0	0	1
GILD	0	0	0	0	0	1	11	ENTG	1	0	0	1	0	0	1
META	0	0	0	0	0	0	0	GEN	0	0	0	0	0	0	0
CMCSA	0	0	0	0	0	0	0	JBHT	0	0	0	0	0	0	0
CTAS	0	0	1	0	0	0	11	NTNX	0	0	0	0	0	0	0
CSCO	0	0	0	0	0	0	0	NTRA	0	1	1	1	0	0	1
FANG	0	0	0	0	1	0	11	UTHR	0	0	0	1	0	0	11
EXC	0	0	0	0	0	0	0	LNT	0	0	0	0	0	1	11
DLTR	0	0	0	0	0	0	0	AKAM	0	0	0	0	0	0	0
CSX	0	0	0	0	0	0	0	DLTR	0	0	0	0	0	0	0
CTSH	0	0	0	0	0	0	0	TRMB	0	0	0	0	0	0	0
DXCM	0	1	0	0	0	0	11	NWS	0	0	0	0	0	0	0
AZN	0	0	0	0	0	1	11	NDSN	0	0	0	0	0	0	0
LIN	0	0	0	0	0	0	0	ENPH	1	1	1	1	0	1	1
CSGP	0	0	1	0	0	0	11	CPB	0	0	0	0	0	1	11
BKNG	0	0	0	0	0	0	0	EVRG	0	0	0	0	0	0	0
CPRT	0	0	1	0	0	0	11								

that were included in the optimal portfolio only twice with code 11 and stocks that were included in the optimal portfolio more than twice with code 1.

To obtain the optimal stock portfolio under the strategy introduced for risky investors, first consider the optimal stock portfolio obtained using the SVM method (high return) in the Table 10. If the stock selected in Table 10 is also among the stocks selected in Tables 13, 14, and 15, then we put it in the final optimal portfolio. However, if the selected stock is not among the stocks selected in Tables 13 or 14 with code 1, we remove it from the optimal portfolio. If a stock is not among the optimal portfolio in Table 10 but is among the optimal portfolios in Tables 13 and 14 with code 1 and Table 15 with code 1 or 11, we add it to the optimal portfolio. By employing this strategy, we successfully identify an optimal stock portfolio in which six out of the thirteen stocks introduced for the year 2023 have been accurately determined (See columns 1 to 6 of Table 16).

To identify a more efficient optimal portfolio for risk-averse investors, we employ a systematic approach. Utilizing Tables 10, 11, and 12, we then determine the optimal stock portfolio. First, we assess the stocks designated in Table 10. If these stocks are also categorized with codes 1 or 11 in Tables 11 and 12, they are included in the optimal stock portfolio. Conversely, stocks that are absent from either Table 11

Table 14. **Categorizing stocks based on their Return.**

	2017	2018	2019	2020	2021	2022	Selected stocks		2017	2018	2019	2020	2021	2022	Selected stocks
PANW	0	1	0	1	1	0	1	CHTR	0	0	1	0	0	0	0
PAYX	0	0	0	0	1	0	0	CDNS	1	0	1	1	1	0	1
MDLZ	0	0	0	0	0	1	0	ADBE	1	1	1	1	0	0	1
ODFL	1	0	1	1	1	0	1	BIIB	0	0	0	0	0	1	0
MCHP	1	0	1	0	0	0	11	AVGO	1	0	0	1	1	0	1
ORLY	0	1	0	0	1	1	1	BKR	0	0	0	0	0	1	0
NXPI	0	0	1	0	1	0	11	STX	0	0	1	0	1	0	11
IDXX	1	1	0	1	0	0	1	PTC	0	1	0	1	0	1	1
KLAC	0	0	1	1	1	0	1	TER	1	0	1	1	1	0	1
KDP	0	0	0	0	0	1	0	CHKP	0	0	0	0	0	1	0
MAR	1	0	0	0	0	0	0	HBAN	0	0	0	0	0	1	0
MRVL	1	0	1	1	1	0	1	COO	0	1	0	0	0	0	0
MELI	1	0	1	1	0	0	1	PFG	0	0	0	0	1	1	11
KHC	0	0	0	0	0	1	0	STLD	0	0	0	0	1	1	11
LULU	0	1	1	1	0	0	1	EXPE	0	0	0	0	1	0	0
LRCX	1	0	1	1	1	0	1	UAL	0	1	0	0	0	0	0
FAST	0	0	1	0	0	0	0	WBD	0	1	0	0	0	0	0
ISRG	1	1	0	1	1	0	1	ZBRA	0	1	1	1	1	0	1
TTD	1	1	1	1	0	0	1	HOLX	0	0	0	1	0	1	11
INTU	1	1	0	1	1	0	1	VRSN	1	1	0	0	0	0	11
FTNT	1	1	1	1	1	0	1	NTRS	0	0	0	0	0	0	0
INTC	0	0	0	0	0	0	0	LPLA	1	0	1	0	1	1	1
HON	0	0	0	0	0	1	0	SSNC	1	1	0	0	0	0	11
ILMN	1	1	0	0	0	0	11	ALGN	1	0	1	1	0	0	1
GILD	0	0	0	0	0	1	0	ENTG	1	0	1	1	1	0	0
META	1	0	1	0	0	0	11	GEN	0	0	0	0	0	0	0
CMCSA	0	0	0	0	0	0	0	JBHT	0	0	0	0	1	0	0
CTAS	0	0	1	0	0	1	11	NTNX	0	1	0	0	0	0	0
CSCO	0	1	0	0	1	0	11	NTRA	0	1	1	1	0	0	1
FANG	0	0	0	0	1	1	11	UTHR	0	0	0	1	1	1	1
EXC	0	1	0	0	1	1	1	LNT	0	0	0	0	0	0	0
DLTR	0	0	0	0	0	1	0	AKAM	0	0	1	0	0	0	0
CSX	1	1	0	0	0	0	11	DLTR	0	0	0	0	0	1	0
CTSH	0	0	0	0	0	0	0	TRMB	1	0	0	1	0	0	11
DXCM	0	1	1	1	1	0	1	NWS	1	0	0	0	0	0	0
AZN	0	1	0	0	0	1	11	NDSN	0	0	0	0	0	1	0
LIN	0	0	0	0	1	1	11	ENPH	1	1	1	1	0	1	1
CSGP	1	1	1	1	0	1	1	CPB	0	0	0	0	0	1	0
BKNG	0	0	0	0	0	0	0	EVRG	0	1	0	0	0	1	11
CPRT	1	1	1	1	0	0	1								

or Table 12 are excluded from the portfolio. Additionally, if any stocks were not selected through the machine learning method but were included in the optimal stock portfolio in both other tables and designated with code 1, these stocks are added to the final optimal stock portfolio. By employing this strategy, we successfully identify an optimal stock portfolio in which six out of the nine stocks introduced for the year 2023 have been accurately determined (See columns 7 to 12 of Table 16).

Although the empirical results presented in this study demonstrate the robustness and accuracy of the hybrid SLV-DGM-machine learning framework for portfolio optimization, several limitations must be acknowledged, particularly when considering real-time market applications. First, the DGM, while mesh-free and scalable, requires offline training that can be computationally intensive. Once trained, the network provides fast option price evaluations; however, market conditions such as volatility regimes, interest rates, and asset correlations can change abruptly. In a real-time setting, the model would need periodic retraining or fine-tuning to remain accurate, which may not be feasible during periods of extreme market stress when computational resources are limited. Second, the long-memory stochastic local volatility (SLV) model depends on the Hurst parameter H (with $3/4 < H < 1$), which captures long-range dependence. Estimating H reliably from historical data is challenging,

Table 15. **Categorizing stocks based on high volatility.**

	2017	2018	2019	2020	2021	2022	Selected stocks		2017	2018	2019	2020	2021	2022	Selected stocks
PANW	1	0	1	1	0	0	1	CHTR	1	0	0	0	0	0	0
PAYX	0	0	0	0	0	0	0	CDNS	0	0	1	0	0	1	11
MDLZ	0	0	0	0	0	0	0	ADBE	1	0	0	0	1	0	11
ODFL	0	1	0	0	0	0	0	BIIB	0	0	1	0	1	1	1
MCHP	0	1	1	1	0	1	1	AVGO	1	1	0	0	0	0	11
ORLY	1	0	0	0	0	0	0	BKR	1	1	0	1	1	1	1
NXPI	0	1	0	1	0	1	1	STX	1	1	1	0	1	0	1
IDXX	0	0	0	0	0	0	0	PTC	0	0	1	0	0	0	0
KLAC	0	1	1	0	0	1	1	TER	1	1	0	0	1	1	1
KDP	0	1	0	0	0	0	0	CHKP	0	0	0	0	0	0	0
MAR	0	0	0	0	1	1	11	HBAN	0	0	0	0	1	0	0
MRVL	1	0	0	1	1	1	1	COO	0	0	0	0	0	0	0
MELI	1	1	1	1	1	1	1	PFG	0	0	0	1	0	0	0
KHC	0	0	1	0	0	0	0	STLD	0	0	1	0	1	1	1
LULU	1	1	0	1	1	1	1	EXPE	0	0	0	1	1	1	1
LRCX	1	1	1	0	0	1	1	UAL	1	0	0	1	1	1	1
FAST	0	1	1	0	0	0	11	WBD	1	1	0	1	1	1	1
ISRG	0	0	0	0	1	1	11	ZBRA	0	1	0	0	1	1	1
TTD	1	1	1	1	1	1	1	HOLX	0	0	0	1	1	0	11
INTU	0	0	0	0	0	0	0	VRSN	0	0	0	0	0	0	0
FTNT	0	1	1	0	0	0	11	NTRS	0	0	0	0	0	0	0
INTC	0	0	0	0	0	0	0	LPLA	0	0	1	1	0	0	11
HON	0	0	0	0	0	0	0	SSNC	0	0	1	1	0	0	11
ILMN	0	1	0	0	1	1	1	ALGN	1	1	1	1	0	1	1
GILD	0	0	0	0	0	0	0	ENTG	1	0	1	1	0	1	1
META	0	0	1	0	0	1	11	GEN	1	1	1	1	1	0	1
CMCSA	0	0	0	0	0	0	0	JBHT	0	0	0	0	0	0	0
CTAS	0	0	1	1	0	0	11	NTNX	1	1	1	1	1	1	1
CSCO	0	0	0	0	0	0	0	NTRA	1	1	1	1	1	1	1
FANG	1	1	0	1	1	1	1	UTHR	1	0	0	0	1	1	1
EXC	0	0	0	0	0	0	0	LNT	0	0	0	0	0	0	0
DLTR	0	0	1	0	1	0	11	AKAM	1	0	0	0	0	0	0
CSX	1	0	0	0	0	0	0	DLTR	0	0	1	0	1	0	11
CTSH	0	0	0	1	0	0	0	TRMB	0	1	0	0	0	0	0
DXCM	1	1	1	0	1	1	1	NWS	0	0	0	1	0	0	0
AZN	0	0	0	0	0	0	0	NDSN	1	0	0	0	0	0	0
LIN	0	0	0	0	0	0	0	ENPH	1	1	1	1	1	1	1
CSGP	0	0	0	0	0	0	0	CPB	1	1	0	0	0	0	11
BKNG	0	0	0	1	1	0	11	EVRG	0	0	0	0	0	0	0
CPRT	0	1	0	0	0	0	0								

and small estimation errors can lead to significant mispricing of options, especially for long-maturity contracts. Third, the machine learning classifiers (Random Forest, KNN, and SVM) were trained on data from 2017 to 2022 and used to predict risk categories for 2023. In real-time deployment, the relationship between features and risk classes may drift over time due to structural breaks, regulatory changes, or unprecedented events (e.g., financial crises, pandemics). The current framework does not incorporate online learning or adaptive mechanisms to update the classifiers incrementally. Fourth, the CCMV optimization model (4.2) assumes that transaction costs O_i and rebalancing costs k_i are known and constant, whereas in real markets these costs can be highly variable and trade-dependent. Finally, the integration of SLV-derived option prices into the expected return term $\mu_i + (R_{call})_i$ assumes that options are liquid and that their market prices reflect the long-memory dynamics accurately. In practice, options on smaller Russell 2000 stocks may suffer from illiquidity and wide bid-ask spreads, which would violate the no-arbitrage assumptions underlying the PDE derivation. Addressing these limitations would require extending the model to include robust parameter estimation, online machine learning, stochastic transaction costs, and liquidity-aware constraints, which we leave for future research.

Table 16. Comparison of the optimal stock portfolio based on the presented strategy and actual market data for 2023.

	Proposed strategy High return	Real data High return		Proposed strategy High return	Real data High return		Proposed strategy Low risk	Real data Low risk		Proposed strategy Low risk	real data Low risk
PANW	1	1	CHTR	0	0	PANW	0	0	CHTR	0	0
PAYX	0	0	CDNS	1	1	PAYX	1	1	CDNS	0	0
MDLZ	0	0	ADBE	1	1	MDLZ	1	1	ADBE	0	0
ODFL	0	0	BIIB	0	0	ODFL	0	0	BIIB	0	0
MCHP	0	0	AVGO	0	1	MCHP	0	0	AVGO	0	0
ORLY	0	0	BKR	0	0	ORLY	0	0	BKR	0	0
NXPI	0	0	STX	0	0	NXPI	0	0	STX	0	0
IDXX	0	0	PTC	0	0	IDXX	0	0	PTC	0	0
KLAC	1	1	TER	0	0	KLAC	0	0	TER	0	0
KDP	0	0	CHKP	0	0	KDP	0	1	CHKP	0	0
MAR	0	0	HBAN	0	0	MAR	0	0	HBAN	1	0
MRVL	0	0	COO	0	0	MRVL	0	0	COO	0	0
MELI	1	1	PFG	0	0	MELI	0	0	PFG	0	0
KHC	0	0	STLD	0	0	KHC	0	0	STLD	0	0
LULU	1	0	EXPE	0	0	LULU	0	0	EXPE	0	0
LRCX	1	1	UAL	0	0	LRCX	0	0	UAL	0	0
FAST	0	0	WBD	0	0	FAST	0	0	WBD	0	0
ISRG	0	0	ZBRA	1	0	ISRG	0	0	ZBRA	0	0
TTD	1	1	HOLX	0	0	TTD	0	0	HOLX	1	0
INTU	0	0	VRSN	0	0	INTU	0	0	VRSN	1	1
FTNT	0	0	NTRS	0	0	FTNT	0	0	NTRS	0	0
INTC	0	1	LPLA	0	0	INTC	0	0	LPLA	0	0
HON	0	0	SSNC	0	0	HON	0	1	SSNC	0	0
ILMN	0	0	ALGN	1	0	ILMN	0	0	ALGN	0	0
GILD	0	0	ENTG	0	0	GILD	0	1	ENTG	0	0
META	0	1	GEN	0	0	META	0	0	GEN	0	0
CMCSA	0	0	JBHT	0	0	CMCSA	0	0	JBHT	0	0
CTAS	0	0	NTNX	0	1	CTAS	0	0	NTNX	0	0
CSCO	0	0	NTRA	1	0	CSCO	0	0	NTRA	0	0
FANG	0	0	UTHR	0	0	FANG	0	0	UTHR	1	1
EXC	0	0	LNT	0	0	EXC	0	1	LNT	1	1
DLTR	0	0	AKAM	0	0	DLTR	0	0	AKAM	0	0
CSX	0	0	DLTR	0	0	CSX	1	1	DLTR	0	0
CTSH	0	0	TRMB	0	0	CTSH	0	0	TRMB	0	0
DXCM	1	0	NWS	0	0	DXCM	0	0	NWS	0	0
AZN	0	0	NDSN	0	0	AZN	1	0	NDSN	0	0
LIN	0	0	ENPH	1	0	LIN	0	0	ENPH	0	0
CSGP	0	0	CPB	0	0	CSGP	0	0	CPB	0	0
BKNG	0	0	EVRG	0	0	BKNG	0	0	EVRG	0	0
CPRT	0	0				CPRT					

5. Conclusion

This paper introduced a novel long-memory stochastic local volatility (SLV) model for pricing European options, leveraging the computational power of the Deep Galerkin Method (DGM). By deriving the governing partial differential equation through a no-arbitrage argument and employing the DGM for its solution, we have developed a framework that effectively captures the complex dynamics of financial markets, including long-range dependence and the leverage effect. Our numerical experiments demonstrate the significant advantages of this approach. The DGM-based solver proves to be both computationally efficient and highly accurate, outperforming traditional advanced methods like Monte Carlo simulation. This efficiency is particularly valuable for practical applications requiring rapid and reliable pricing. Beyond theoretical pricing, we successfully demonstrated the practical utility of our model in a holistic portfolio optimization workflow. By integrating machine learning for stock selection with the Cardinality-Constrained Mean–Variance (CCMV) model informed by our SLV pricing framework, we constructed optimal portfolios tailored to specific risk tolerances. This empirical application, using historical data from the Russell 2000 index, underscores the model's robustness and its direct relevance to real-world investment decision-making.

Acknowledgments: This research was funded by National Science, Research and In-

novation Fund (NSRF), and King Mongkut's University of Technology North Bangkok with Contract no. KMUTNB-FF-69-B-37.

Conflict of interest: The author declares that there is no Conflict of interest in this paper.

Competing Interests: The author has no relevant financial or non-financial interests to disclose.

Author Contributions: The author contributed to the study conception and design.

Data Availability: The datasets generated during the current study are available.

References

[1] A. Al-Aradi, A. Correia, G. Jardim, D. de Freitas Naiff, and Y. Saporito, *Extensions of the deep Galerkin method*, Appl. Math. Comput. **430** (2022), 127287. https://doi.org/10.1016/j.amc.2022.127287.

[2] M. Bansal, A. Goyal, and A. Choudhary, *A comparative analysis of k-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning*, Decision Analytics Journal **3** (2022), 100071.
https://doi.org/10.1016/j.dajour.2022.100071.

[3] C. Beck, S. Becker, P. Cheridito, A. Jentzen, and A. Neufeld, *Deep splitting method for parabolic PDEs*, SIAM J. Sci. Comput. **43** (2021), no. 5, A3135–A3154.
https://doi.org/10.1137/19M1297919.

[4] L. Boccardo, A. Dall Aglio, T. Gallouet, and L. Orsina, *Nonlinear parabolic equations with measure data*, J. Func. Anal. **147** (1997), no. 1, 237–258.

[5] L. Breiman, *Random forests*, Machine Learning **45** (2001), no. 1, 5–32.
https://doi.org/10.1023/A:1010933404324.

[6] N. Deng, Y. Tian, and C. Zhang, *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions*, CRC Press, 2012.

[7] J. Han and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Commun. Math. Stat. **5** (2017), no. 4, 349–380.

[8] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. **115** (2018), no. 34, 8505–8510.
https://doi.org/10.1073/pnas.1718942115.

[9] Y. He, P. Chen, L. He, K. Xiang, and C. Wu, *A dynamic Heston local–stochastic volatility model and Legendre transform dual-asymptotic solution for optimal investment strategy problems with CARA utility*, J. Comput. Appl. Math. **423**

(2023), 114993.
https://doi.org/10.1016/j.cam.2022.114993.

[10] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Comput. **9** (1997), no. 8, 1735–1780.

[11] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Netw. **4** (1991), no. 2, 251–257.
https://doi.org/10.1016/0893-6080(91)90009-T.

[12] C. Hure, H. Pham, and X. Warin, *Deep backward schemes for high-dimensional nonlinear PDEs*, Math. Comp. **89** (2020), no. 324, 1547–1579.
https://doi.org/10.1090/mcom/3514.

[13] M. Hutzenthaler, A. Jentzen, T. Kruse, and T.A. Nguyen, *A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations*, SN Partial Differential Equations and Applications **1** (2020), no. 2, 10.
https://doi.org/10.1007/s42985-019-0006-9.

[14] D. Kim, M. Ha, J.H. Kim, and J. H. Yoon, *A local volatility correction to mean-reverting stochastic volatility model for pricing derivatives*, The Quarterly Review of Economics and Finance **97** (2024), 101901.
https://doi.org/10.1016/j.qref.2024.101901.

[15] H.G. Kim and J.H. Kim, *A stochastic-local volatility model with Lévy jumps for pricing derivatives*, Appl. Math. Comput. **451** (2023), 128034.
https://doi.org/10.1016/j.amc.2023.128034.

[16] O.A. Ladyzhenskaia, V.A. Solonnikov, and N.N. Ural'tseva, *Linear and Quasi-Linear Equations of Parabolic Type*, vol. 23, American Mathematical Soc., 1968.

[17] J. Ma, W. Yang, and Z. Cui, *Convergence analysis for continuous-time Markov chain approximation of stochastic local volatility models: option pricing and greeks*, J. Comput. Appl. Math. **404** (2022), 113901.
https://doi.org/10.1016/j.cam.2021.113901.

[18] A. Najafi and F. Mehrdoust, *Conditional expectation strategy under the long memory Heston stochastic volatility model*, Communications in Statistics-Simulation and Computation **53** (2024), no. 11, 5453–5473.
https://doi.org/10.1080/03610918.2023.2189165.

[19] S. Pagliarani and A. Pascucci, *Local stochastic volatility with jumps: analytical approximations*, Int. J. Theor. Appl. Finance **16** (2013), no. 08, 1350050.
https://doi.org/10.1142/S0219024913500507.

[20] M.M. Porzio, *Existence of solutions for some "noncoercive" parabolic equations*, Discrete Contin. Dyn. Syst. **5** (1999), 553–568.
https://doi.org/10.3934/dcds.1999.5.553.

[21] Y.F. Saporito, X. Yang, and J.P. Zubelli, *The calibration of stochastic local-volatility models: An inverse problem perspective*, Computers and Mathematics with Applications **77** (2019), no. 12, 3054–3067.
https://doi.org/10.1016/j.camwa.2019.01.029.

[22] K. Shiraya and A. Takahashi, *An approximation formula for basket option prices under local stochastic volatility with jumps: An application to commodity markets*,

J. Comput. Appl. Math. **292** (2016), 230–256.
https://doi.org/10.1016/j.cam.2015.06.027.

[23] J. Simon, *Compact sets in the space $l^p(o,t;b)$*, Ann. Mat. Pura Appl. **146** (1986), no. 1, 65–96.

[24] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys. **375** (2018), 1339–1364.
https://doi.org/10.1016/j.jcp.2018.08.029.

[25] R.K. Srivastava, K. Greff, and J. Schmidhuber, *Highway networks*, 2015.
https://arxiv.org/abs/1505.00387.

[26] G. Xu and H. Zheng, *Basket options valuation for a local volatility jump–diffusion model with the asymptotic expansion method*, Insurance: Mathematics and Economics **47** (2010), no. 3, 415–422.
https://doi.org/10.1016/j.insmatheco.2010.08.008.