TEMPORAL DIFFERENCE LEARNING FOR HIGH-DIMENSIONAL PIDEs WITH JUMPS*

LIWEI LU†, HAILONG GUO‡, XU YANG§, AND YI ZHU¶

Abstract. In this paper, we propose a deep learning framework for solving high-dimensional partial integro-differential equations (PIDEs) based on the temporal difference learning. We introduce a set of Lévy processes and construct a corresponding reinforcement learning model. To simulate the entire process, we use deep neural networks to represent the solutions and nonlocal terms of the equations. Subsequently, we train the networks using the temporal difference error, the termination condition, and properties of the nonlocal terms as the loss function. The relative error of the method reaches $\mathcal{O}(10^{-3})$ in 100-dimensional experiments and $\mathcal{O}(10^{-4})$ in one-dimensional pure jump problems. Additionally, our method demonstrates the advantages of low computational cost and robustness, making it well-suited for addressing problems with different forms and intensities of jumps.

Key words. Lévy motion, reinforcement learning, partial integro-differential equation, temporal difference learning

MSC codes. 34K30, 60H35, 60J76, 68T07

DOI. 10.1137/23M1584538

1. Introduction. The partial integro-differential equation (PIDE) has found widespread applications in various real-world scenarios, including engineering, biology, and particularly finance [1, 13, 18]. Examples of such phenomena can be observed in ohmic heating, resistance spot welding, the Gierer–Meinhardt system, and option pricing [13, 18]. Classical methods such as the finite difference method [6, 21] and the finite element method [14, 29] have traditionally been used to solve PIDEs, and they exhibit good performance in low-dimensional problems. However, these mesh-based approaches face significant challenges in high-dimensional cases due to the curse of dimensionality [20]. The computational complexity grows exponentially as the number of dimensions increases, rendering these methods impractical. Nevertheless, many real-world models formulated as PIDEs inherently possess high-dimensional characteristics. For instance, in option pricing, the dimensions correspond to the number of underlying assets, while in reaction-diffusion systems of cell dynamics, they correspond to the number of reacting substances.

†Yau Mathematical Sciences Center, Tsinghua University, Beijing, 100084, China (llw20@mails.tsinghua.edu.cn).

‡School of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3010, Australia (hailong.guo@unimelb.edu.au).

§Department of Mathematics, University of California, Santa Barbara, CA 93106 USA (xuyang@math.ucsb.edu).

¶Corresponding author. Yau Mathematical Sciences Center, Tsinghua University, Beijing, 100084, China, and Beijing Institute of Mathematical Sciences and Applications, Beijing, 101408, China (yizhu@tsinghua.edu.cn).

Neural networks have emerged as a promising tool for alleviating the curse of dimensionality in solving high-dimensional partial differential equations (PDEs) without nonlocal terms. In recent years, several neural network–based methods have been widely adopted for this purpose [3, 7]. The physics-informed neural network (PINN) [31, 5, 17, 19, 22, 25, 27, 28, 34, 35] incorporates the residual term of the PDE into the training process and utilizes automatic differentiation to handle the differential operations, and it has sparked a wave of using deep learning to solve PDEs. The idea of the deep Galerkin method (DGM) [32] is similar to PINN, but it incorporates the norm of the residual term in its loss function and adopts an L_2 norm. The deep mixed residual method (MIM) [26] rewrites high-order PDEs into a system of first-order PDEs for solving purposes. The weak adversarial network (WAN) [36] leverages the analogy between the weak form of the PDE and the generative adversarial network, integrating by parts to transfer derivatives onto the test functions. The deep Ritz method [10] utilizes the Ritz variational form to solve high-dimensional PDEs and eigenvalue problems. The deep Nitsche method [24] represents the trial functions by neural networks and employs the Nitsche variational form to handle mixed boundary value problems. In addition, introducing stochastic differential equations (SDEs) related to the PDE provides another potential avenue for mitigating the curse of dimensionality. The deep backward stochastic differential equation (deep BSDE) [15, 16] approximates the gradients of the solution at different time points using a series of neural networks and trains them with the terminal condition. The forward-backward stochastic neural network (FBSNN) [30] incorporates trajectory information into the training process to directly obtain the solution of the equation. The authors of [37] use two neural networks to approximate the solution and its gradient, or by autodifferentiation and performing training at each time point based on a temporal difference method.

When it comes to PIDEs, the literature on deep neural network–based approaches is not as extensive. The authors of [12] extend the work of [15, 16] to a more general case involving jump processes and that uses additional neural networks to approximate nonlocal terms. For problems with a finite number of jumps, the authors of [11] circumvent the computation of nonlocal terms by introducing a stochastic process with integration on a Poisson random measure. In [4], an error estimation technique is proposed for approximating forward-backward stochastic systems using neural networks. Furthermore, the authors of [33] approximate the initial value function, gradients of the solution, and the integral kernel using neural networks, with the termination condition serving as a constraint.

This paper aims to numerically compute solutions to PIDEs. A group of Lévy-type forward-backward stochastic processes is introduced based on the target PIDE. Inspired by [37], a reinforcement learning framework is established. The equation's solution and nonlocal terms are represented using neural networks. Subsequently, a loss function is constructed to update the network parameters, taking into account the errors from temporal difference methods, termination conditions, and the properties of nonlocal terms. This method exhibits two primary advantages: (1) It exhibits a reduced computational cost. Capitalizing on the benefits of temporal difference learning, the proposed approach eliminates the need to wait for the completion of an entire trajectory simulation before updating parameters. Moreover, as the dimensionality increases, the required number of trajectories does not experience significant growth. (2) The method has fast convergence and high precision. With rapid convergence, the error remains within the range of $\mathcal{O}(10^{-4})$ for one-dimensional pure jump problems and $\mathcal{O}(10^{-3})$ for high-dimensional problems.

The rest of this paper is organized as follows. Section 2 presents the overall methodology, which involves the incorporation of Lévy processes and reinforcement learning models to establish the training objectives. Section 3 demonstrates numerical examples for solving one-dimensional and high-dimensional PIDEs. Finally, in section 4, we provide a summary of the entire approach.

2. Methodology. The Brownian motion, defined based on the Gaussian distribution, has been widely adopted for modeling noise. However, in the scenarios of real applications, non-Gaussian noises are better characterized by Lévy motion, which is particularly relevant in fields like finance, chemistry, engineering, and geophysics [8, 9]. Lévy processes are stochastic processes that satisfy zero initial values, stationary independent increments, and stochastically continuous sample paths [2]. For any Lévy process L_t, the Poisson random measure N, the compensated Poisson random measure $\widetilde{N}$, and the jump measure ν can be defined by

$$(2.1) \qquad N(t,S)(\omega) := \operatorname{card}\{s \in [0,t) : L_s(\omega) - L_{s-}(\omega) \in S\},$$

$$(2.2) \qquad \nu(S) := \mathbb{E}N(1,S)(\omega),$$

$$(2.3) \qquad \widetilde{N}(t,S) = N(t,S) - t\nu(S),$$

where S is a Borel set on $\mathbb{R}^d \backslash \{0\}$.

The Lévy form of the Itô's formula bridges the gap between integro-differential operators and Lévy processes. Nevertheless, there remains a considerable disparity in simulating the Lévy process and accurately computing the nonlocal terms in the operator. Furthermore, efficiently solving high-dimensional problems at a lower cost presents a formidable challenge. To tackle these issues, we propose a framework that employs reinforcement learning and utilizes deep neural networks.

2.1. PIDE and Lévy process. In the present investigation, our attention is directed towards the resolution of the following partial integro-differential equation (PIDE):

$$(2.4) \qquad \begin{cases} \dfrac{\partial u}{\partial t} + b \cdot \nabla u + \dfrac{1}{2}\operatorname{Tr}(\sigma\sigma^T \mathrm{H}(u)) + \mathcal{A}u + f = 0, \\ u(T,\cdot) = g(\cdot), \end{cases}$$

where ∇u and $\mathrm{H}(u)$ correspond to the gradient and Hessian matrix of the function $u(t,x)$ with respect to the spatial variable $x \in \mathbb{R}^d$, and $0 < t < T$. The vectors $b = b(x) \in \mathbb{R}^d$, the matrix $\sigma = \sigma(x) \in \mathbb{R}^{d \times d}$, and the function $f = f(t,x,u,\sigma^T\nabla u) \in \mathbb{R}$, are all predefined. Here

$$(2.5) \qquad \mathcal{A}u(t,x) = \int_{\mathbb{R}^d} (u(t,x+G(x,z)) - u(t,x) - G(x,z) \cdot \nabla u(t,x))\nu(\mathrm{d}z),$$

where $G = G(x,z) \in \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$, and ν is a Lévy measure associated with a Poisson random measure N.

Traditional approaches, such as finite element and finite difference methods, tend to be insufficient in dealing with nonlocal terms and high-dimensional scenarios. To overcome these limitations, we propose the following Lévy-type process on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ in the Itô sense:

$$(2.6) \qquad \mathrm{d}X_t = b(X_t)\,\mathrm{d}t + \sigma(X_t)\,\mathrm{d}W_t + \int_{\mathbb{R}^d} G(X_t,z)\widetilde{N}(\mathrm{d}t,\mathrm{d}z).$$

In this context, $\{W_t\}_{t=0}^{T}$ denotes a d-dimensional Brownian motion, and $\widetilde{N}(\mathrm{d}t, \mathrm{d}z) = N(\mathrm{d}t, \mathrm{d}z) - \nu(\mathrm{d}z)\,\mathrm{d}t$ represents the compensated Poisson measure. $\{\mathcal{F}_t\}_{t=0}^{T}$ is the filtration generated by Brownian motion W_t and Poisson random measure N. If X_0 is assumed to follow a known single-point distribution $X_0 = \xi \in \mathbb{R}^d$, the entire process X_t can be completely known and simulated. Subsequently, we introduce the following three stochastic processes:

$$(2.7) \quad Y_t = u(t, X_t), \ Z_t = \nabla u(t, X_t), \ U_t = \int_{\mathbb{R}^d} (u(t, X_t + G(X_t, z)) - u(t, X_t))\nu(\mathrm{d}z).$$

According to the Itô formula for Lévy-type stochastic integrals [2],

$$
\begin{aligned}
\mathrm{d}Y_t = & \left[\frac{\partial u}{\partial t}(t, X_t) + b(X_t) \cdot \nabla u(t, X_t) \right. \\
& \left. + \frac{1}{2}\mathrm{Tr}\left(\sigma(X_t)\sigma(X_t)^T \mathrm{H}(u)(t, X_t)\right) + \mathcal{A}u(t, X_t) \right] \mathrm{d}t \\
& + \nabla u(t, X_t) \cdot \sigma(X_t)\,\mathrm{d}W_t + \int_{R^d} [u(t, X_t + G(X_t, z)) - u(t, X_t)]\,\widetilde{N}(\mathrm{d}t, \mathrm{d}z) \\
= & - f(t, X_t, Y_t, \sigma(X_t)^T Z_t)\,\mathrm{d}t + (\sigma(X_t)^T Z_t)^T\,\mathrm{d}W_t \\
& + \int_{R^d} [u(t, X_t + G(X_t, z)) - u(t, X_t)]\,\widetilde{N}(\mathrm{d}t, \mathrm{d}z),
\end{aligned}
$$

(2.8)

with $Y_T = g(X_T)$.

Equations (2.6) and (2.8) are collectively referred to as the forward-backward stochastic differential equation (FBSDE) system [30], which establishes a fundamental link between solving the equation and simulating the stochastic process. In other words, if $u(t, x)$ represents a solution to (2.4), the processes (X_t, Y_t, Z_t, U_t) defined through (2.6) and (2.7) are required to satisfy (2.8). This provides a foundation for assessing the accuracy of the approximate solution $u(t, x)$.

In order to simulate these stochastic processes, the time $[0, T]$ is divided into N equal subintervals: $0 = t_0 < t_1 < t_2 < \cdots < t_N = T$, resulting in the numerical discretization of (2.6)

$$
\begin{aligned}
X_{t_{n+1}} = & X_{t_n} + b(X_{t_n})\Delta t + \sigma(X_{t_n})\Delta W_n \\
& + \sum_{i=N_n+1}^{N_{n+1}} G(X_{t_n}, z_i) - \Delta t \int_{\mathbb{R}^d} G(X_{t_n}, z)\nu(\mathrm{d}z)
\end{aligned}
$$

(2.9)

and the discretization of (2.8)

$$
\begin{aligned}
Y_{t_{n+1}} = & Y_{t_n} - f(t_n, X_{t_n}, Y_{t_n}, \sigma^T Z_{t_n})\Delta t + (\sigma(t_n, X_{t_n})^T Z_{t_n})^T \Delta W_n \\
& + \sum_{i=N_n+1}^{N_{n+1}} [u(t_n, X_{t_n} + G(X_{t_n}, z_i)) - u(t_n, X_{t_n})] - \Delta t U_{t_n}.
\end{aligned}
$$

(2.10)

The terms ΔW_n in both (2.9) and (2.10) correspond to a same sample drawn from normal distribution $\mathcal{N}(0, \Delta t)$. N_n represents the number of jumps that occur in the interval $[0, t_n]$ for the Poisson random measure N, while $z_i \in \mathbb{R}^d$ denotes the size of the ith jump if arranging all jumps in the interval $[0, T]$ in chronological order. This discretization provides a practical framework for numerical approximation.

2.2. Neural network approximations. The discretized versions of (2.9) and (2.10) offer a viable approach for simulating the stochastic processes. However, several aspects remain unclear and require further investigation before achieving the final solution. While the simulation of the stochastic process X_t can proceed smoothly once the initial value $X_0 = \xi$ is determined, the simulation of Y_t presents greater challenges due to its dependence on Y_t itself, as well as Z_t and U_t. The computation of the integral form in U_t is a matter that needs careful consideration, and accurately calculating the difference in Z_t is also not a trivial task.

In this work, neural networks have been adopted as an approximation for the solution $u(t, x)$ of the PIDE (2.4). Specifically, when the values of X_t are known, they can be readily fed into the neural network to obtain an approximation of Y_t. Moreover, the automatic differentiation technique enables us to conveniently approximate Z_t. However, the computation of U_t remains a challenging task. Employing Monte Carlo methods to evaluate the integral in U_t might not be an optimal choice, as it necessitates repetitive calculations for every possible value of X_t and would require recomputation whenever $u(t, x)$ undergoes changes.

To address the challenges associated with the computation of U_t, a modification is made to the output of the neural network. Specifically, two components are introduced: one output $\mathcal{N}_1$ to represent $u(t, x)$ and the other output $\mathcal{N}_2$ to represent $\int_{\mathbb{R}^d}(u(t, x + G(x, z)) - u(t, x))\nu(\mathrm{d}z)$. It is important to note that the value of $\mathcal{N}_2$ depends not only on the input (t, x) but also on the current $u(t, x)$. Consequently, employing a neural network with two outputs may be more suitable than using two separate networks, as it effectively captures the inherent relationship between these outputs. The structure of the neural network is depicted in Figure 1, and it can be mathematically described as

$$(2.11) \quad \begin{cases} \boldsymbol{a}^1 = \boldsymbol{W}^1\boldsymbol{a}^0 + \boldsymbol{b}^1, \quad \boldsymbol{a}^0 = (t, x) \in \mathbb{R}^{d+1}, \\ \boldsymbol{a}^i = h(\boldsymbol{W}^{2i-1}h(\boldsymbol{W}^{2i-2}\boldsymbol{a}^{i-1} + \boldsymbol{b}^{2i-2}) + \boldsymbol{b}^{2i-1}) + \boldsymbol{a}^{i-1}, \quad i = 2, 3, \ldots, l, \\ \boldsymbol{a}^{l+1} = \boldsymbol{W}^{2l}\boldsymbol{a}^l + \boldsymbol{b}^{2l} \in \mathbb{R}^2. \end{cases}$$

Here the $(d + 1)$-dimensional input (t, x) undergoes a linear layer, several residual blocks, and another linear layer, resulting in a two-dimensional output. Each residual block consists of two linear layers combined with an activation function h, along with a residual connection.
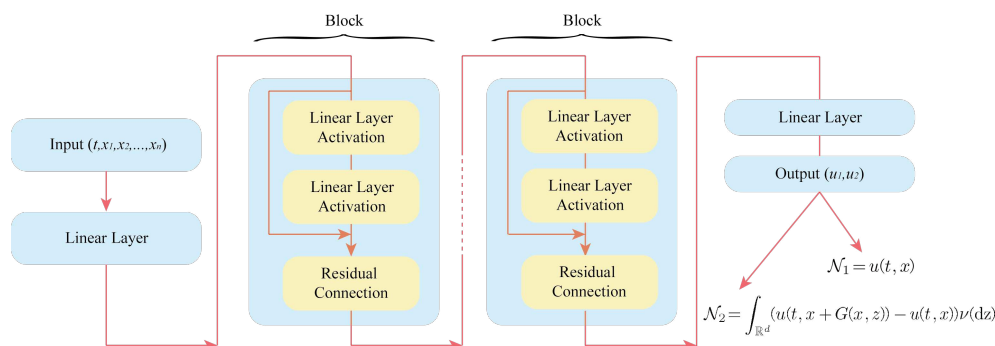

FIG. 1. *Architecture of the residual network utilized in this work. The neural network takes a $(d + 1)$-dimensional input consisting of time $t \in \mathbb{R}$ and spatial variable $x \in \mathbb{R}^d$. The two outputs represent $u(t, x)$ and $\int_{\mathbb{R}^d}(u(t, x + G(x, z)) - u(t, x))\nu(\mathrm{d}z)$, respectively.*

Upon utilizing a neural network as an approximation,

$$
\begin{aligned}
Y_{t_{n+1}} = {} & Y_{t_n} - f(t_n, X_{t_n}, \mathcal{N}_1(t_n, X_n), \sigma^T \nabla \mathcal{N}_1(t_n, X_n)) \Delta t \\
& + (\sigma(t_n, X_{t_n})^T \nabla \mathcal{N}_1(t_n, X_n))^T \Delta W_n \\
& + \sum_{i=N_n+1}^{N_{n+1}} [\mathcal{N}_1(t_n, X_{t_n} + G(X_{t_n}, z_i)) - \mathcal{N}_1(t_n, X_{t_n})] - \Delta t \mathcal{N}_2(t_n, X_{t_n})
\end{aligned}
\tag{2.12}
$$

allows us to simulate Y_t, where the definition of z_i and N_n remain consistent with (2.10). Consequently, it becomes necessary to impose a constraint on $\mathcal{N}_2$. There are various methods to enforce this constraint, and in this study, it is observed that

$$
\left\{ \int_0^t \int_{\mathbb{R}^d} (u(t, X_t + G(X_t, z)) - u(t, X_t)) \widetilde{N}(\mathrm{d}t, \mathrm{d}z) \right\}_{t=0}^T
\tag{2.13}
$$

behaves as a martingale, implying

$$
\mathbb{E}\left[\int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^d} u(t, X_t + G(X_t, z)) - u(t, X_t) \widetilde{N}(\mathrm{d}t, \mathrm{d}z) \Big| \mathcal{F}_{t_n} \right] = 0
\tag{2.14}
$$

within any time $[t_n, t_{n+1}] \subset [0, T]$. Inspired from this observation, we incorporate

$$
\left| \frac{1}{M} \sum_{j=1}^M \left(\sum_{i=N_n+1}^{N_{n+1}} [\mathcal{N}_1(t_n, x_{t_n}^j + G(x_{t_n}^j, z_i^j)) - \mathcal{N}_1(t_n, x_{t_n}^j)] - \Delta t \mathcal{N}_2(t_n, x_{t_n}^j) \right) \right|
\tag{2.15}
$$

into the final loss function, where M denotes the number of training samples, and $x_{t_n}^j$ refers to the jth sample of X_{t_n}. The constraint expressed in (2.15) establishes a link between $\mathcal{N}_1$ and $\mathcal{N}_2$. Consequently, our subsequent objective is to identify a criterion that enables $\mathcal{N}_1$ to closely approximate the solution $u(t, x)$ of the PIDE (2.4).

2.3. Temporal difference learning. As an approach for learning through interaction, reinforcement learning (RL) has gained increasing attention in various fields, including solving high-dimensional PDEs [23]. Temporal difference (TD) is a reinforcement learning method with the form of bootstrapping. It offers the advantage of not requiring prior knowledge about the environment and allows for learning from each state transition without waiting for the completion of a full trajectory. Remarkably, (2.10) delineates the process Y_t in an incremental manner, aligning harmoniously with the fundamental principles of TD learning.

This observation leads to the establishment of an RL model for our problem by defining a triplet (S, R, P). The state $S = [0, T] \times \mathbb{R}^d$ encompasses the set of all possible spatiotemporal pairs $s = (t, x)$ under the initial condition $X_0 = \xi$. The transition probability

$$
P_{s_1 s_2} = \mathbb{P}(S_2 = (t_2, x_2) | S_1 = (t_1, x_1)) = \mathbb{P}(X_{t_2} = x_2 | X_{t_1} = x_1)
\tag{2.16}
$$

represents the probability of transitioning from x_1 at time t_1 to x_2 at time t_2. Since the state process X_t fully determines the transition probability P, the specific values of P can be calculated using (2.9), making it a model-based problem. However, we will assume that this remains a model-free problem, as our approach does not require knowledge of the transition probability P. Thus, our method can be generalized to more scenarios.

Furthermore, we aim to define a reward R such that the corresponding state value function of the model aligns precisely with the solution $u(t,x)$ of PIDE (2.4). This allows us to transform the solution of (2.4) into the computation of the value function within the framework of RL. If $u(t,x)$ is employed as the value function in an RL model, the process Y_t defined in (2.10) corresponds to the cumulative reward associated with the state process X_t. Hence, the reward can be defined by

$$R_{s_n s_{n+1}} = -f(t_n, X_{t_n}, Y_{t_n}, \sigma^T Z_{t_n})\Delta t + (\sigma(X_{t_n})^T Z_{t_n})^T \Delta W_n$$

(2.17)

$$+ \sum_{i=N_n+1}^{N_{n+1}} [u(t_n, X_{t_n} + G(X_{t_n}, z_i)) - u(t_n, X_{t_n})] - \Delta t U_{t_n},$$

using the incremental formulation in (2.10) for Y_t. And in the case of neural network approximation, it can be computed by

(2.18)

$$R_{s_n s_{n+1}} = -f(t_n, X_{t_n}, \mathcal{N}_1(t_n, X_n), \sigma^T \nabla \mathcal{N}_1(t_n, X_n))\Delta t$$
$$+ (\sigma(X_{t_n})^T \nabla \mathcal{N}_1(t_n, X_n))^T \Delta W_n$$
$$+ \sum_{i=N_n+1}^{N_{n+1}} [\mathcal{N}_1(t_n, X_{t_n} + G(X_{t_n}, z_i)) - \mathcal{N}_1(t_n, X_{t_n})] - \Delta t \mathcal{N}_2(t_n, X_{t_n}).$$

TD learning updates the current value function at each time step without requiring the completion of the entire trajectory over the time interval $[0,T]$. Therefore, in this problem, the loss function is computed and optimization is performed at each time step. Since the current model does not involve actions and policies,

(2.19)
$$u(s_n) \leftarrow u(s_n) + \alpha(R_{s_n s_{n+1}} + u(s_n) - u(s_{n+1}))$$

is utilized to update the value function $u(t,x)$ in the classical temporal difference method. The TD error $R_{s_n s_{n+1}} + u(s_n) - u(s_{n+1})$ serves as a measure of the accuracy of the current value function, which can be effectively used to impose constraints on the neural network $\mathcal{N}_1$. Consequently, the TD error is incorporated as the first component of the loss function

(2.20)
$$\text{TD\_Error}_{t_n}^j = -f(t_n, x_{t_n}^j, \mathcal{N}_1(t_n, x_n^j), \sigma^T \nabla \mathcal{N}_1(t_n, x_n^j))\Delta t$$
$$+ (\sigma(x_{t_n}^j)^T \nabla \mathcal{N}_1(t_n, x_n^j))^T \Delta W_n$$
$$+ \sum_{i=N_n+1}^{N_{n+1}} [\mathcal{N}_1(t_n, x_{t_n}^j + G(x_{t_n}^j, z_i^j)) - \mathcal{N}_1(t_n, x_{t_n}^j)]$$
$$- \Delta t \mathcal{N}_2(t_n, x_{t_n}^j) + \mathcal{N}_1(t_n, x_{t_n}^j) - \mathcal{N}_1(t_{n+1}, x_{t_{n+1}}^j),$$

(2.21)
$$\text{Loss}_{t_n}^1 = \frac{1}{M} \sum_{j=1}^{M} |\text{TD\_Error}_{t_n}^j|^2.$$

Here M represents the number of samples, and (2.20) represents the TD error of the jth sample at time t_n.

Next, since the cumulative reward process Y_t needs to satisfy not only (2.10) but also $Y_T = g(X_T)$, the termination conditions and their gradients are incorporated as the second and third components of the loss function, respectively, and evenly distributed across each time step $0 = t_0 < t_1 < t_2 < \cdots < t_N = T$,

$$(2.22) \qquad \text{Loss}_{t_n}^2 = \frac{1}{N}\frac{1}{M}\sum_{j=1}^{M}|\mathcal{N}_1(T,x_T^j) - g(x_T^j)|^2,$$

$$(2.23) \qquad \text{Loss}_{t_n}^3 = \frac{1}{N}\frac{1}{M}\sum_{j=1}^{M}|\nabla\mathcal{N}_1(T,x_T^j) - \nabla g(x_T^j)|^2.$$

It should be noted that in TD learning, which does not require the simulation of the entire trajectory, the loss function is computed at each time step. Hence, the terminal state X_T remains unknown when $t < T$. To address this challenge, the terminal state X_T from the previous iteration is utilized in the calculation of (2.22) and (2.23), while for the initial iteration, the terminal state is randomly generated.

Last, as discussed in subsection 2.2, (2.15) is incorporated as the fourth component of the loss function:

$$(2.24)$$
$$\text{Loss}_{t_n}^4$$
$$= \left| \frac{1}{M}\sum_{j=1}^{M}\left(\sum_{i=N_n+1}^{N_{n+1}} [\mathcal{N}_1(t_n, x_{t_n}^j + G(x_{t_n}^j, z_i^j)) - \mathcal{N}_1(t_n, x_{t_n}^j)] - \Delta t \mathcal{N}_2(t_n, x_{t_n}^j) \right) \right|.$$

It serves to constrain the output of the neural network $\mathcal{N}_2$, aiming to make it as close to $\int_{\mathbb{R}^d}(\mathcal{N}_1(t, x+G(x,z)) - \mathcal{N}_1(t,x))\nu(\mathrm{d}z)$ as possible. Based on these, the loss function at time step t_n can be defined in the following manner:

$$(2.25) \qquad \text{Loss}_{t_n} = \text{Loss}_{t_n}^1 + \text{Loss}_{t_n}^2 + \text{Loss}_{t_n}^3 + \text{Loss}_{t_n}^4.$$

The pseudocode for the entire method is presented in Algorithm 2.1. The overall process of this approach is illustrated in Figure 2. In summary, the solution to the PIDE (2.4) is sought by introducing a set of Lévy-type forward-backward stochastic processes (X_t, Y_t, Z_t, U_t). The simulation of Y_t involves unknown functions $u(t,x)$ and $\int_{\mathbb{R}^d}(u(t, x + G(x,z)) - u(t,x))\nu(\mathrm{d}z)$, which are represented separately using a residual network with two outputs. An RL framework is then established, where X_t corresponds to the state process, the solution $u(t,x)$ of (2.4) corresponds to the value function, Y_t corresponds to the cumulative reward process, and the reward is obtained from (2.10). By employing TD learning and termination conditions and leveraging the properties of martingales, the final loss function is constructed. Finally, the parameters of the neural network are updated using optimization methods such as Adam. Through these steps, the solution to (2.4) is successfully obtained.

Remark 2.1. From an alternative perspective, (2.19) employs a one-step TD learning, where after the state transitions from s_n to s_{n+1}, the sum of the reward $R_{s_n s_{n+1}}$ and $u(s_n)$ is used as an approximation for $u(s_{n+1})$. However, it is also possible to employ a two-step TD learning

$$(2.26) \qquad u(s_n) \leftarrow u(s_n) + \alpha(R_{s_n s_{n+1}} + R_{s_{n+1}s_{n+2}} + u(s_n) - u(s_{n+2})),$$

whereby the state progresses from s_n to s_{n+2}, and the sum of $R_{s_n s_{n+1}} + R_{s_{n+1}s_{n+2}}$ and $u(s_n)$ is utilized to approximate $u(s_{n+2})$. Consequently, the TD error and loss1 would undergo corresponding adjustments. If, during a single iteration where the samples run from t_0 to t_N, the one-step TD learning optimizes the parameters $2m$ times, then the two-step TD learning optimizes the parameters m times since it updates them every two steps. Similarly, this approach can be generalized to a k-step temporal difference learning.

Algorithm 2.1. TD learning on high-dimensional PIDEs.

Input: PIDE (2.4), neural network with two outputs $(\mathcal{N}_1, \mathcal{N}_2)$ and parameter θ,
　　snapshots $0 = t_0 < t_1 < t_2 < \cdots < t_N = T$, sample number M, maximum iterations
　　Iterations, initial distribution $X_0 = \xi$, learning rate α.
Output: Approximate solution $\mathcal{N}_1$ of PIDE (2.4).
 1: Initialize neural network. *iteration* $= 0$. Randomly generate M terminal state
　　$\{x_T^j\}_{j=1}^M$ for the initial iteration. Initialize the previous terminal state
　　$\mathbb{X} = \{x_T^j\}_{j=1}^M$.
 2: **while** *iteration* $\leq$ *Iterations* **do**
 3:　　Sample M Brownian motion $\{\{\Delta W_n^j\}_{n=1}^N\}_{j=1}^M$ for each time interval $[t_n, t_{n+1}]$.
　　　Sample some jumps $\{z_i^j\}_{j=1}^M$ for each trajectory j from Lévy motion (see
　　　section 3 for details).
 4:　　**for** time step $n = 0 \to N - 1$ **do**
 5:　　　Update state (t_n, x_n^j) to (t_{n+1}, x_{n+1}^j) by (2.9), $j = 1, 2, \ldots, M$.
 6:　　　Compute $\mathrm{Loss}_{t_n}^1$ by (2.21) and $\mathrm{Loss}_{t_n}^4$ by (2.24).
 7:　　　**if** $n = N - 1$ **then**
 8:　　　　Update the previous terminal state $\mathbb{X} = \{x_T^j\}_{j=1}^M$.
 9:　　　**end if**
10:　　　Compute $\mathrm{Loss}_{t_n}^2$ by (2.22) and $\mathrm{Loss}_{t_n}^3$ by (2.23) using the current $\mathbb{X}$.
11:　　　Compute Loss_{t_n} by (2.25). Optimize the parameter θ of the neural network.
12:　　**end for**
13:　　*iteration* $=$ *iteration* $+ 1$
14: **end while**

---

3. Numerical results. To facilitate the generation of the jump sizes z_i in (2.9)
and (2.10), we consider the Lévy measure $\nu(\mathrm{d}z)$ in the form of $\nu(\mathrm{d}z) = \lambda\phi(z)\,\mathrm{d}z$. Here
$\phi(z)$ denotes the density function of a d-dimensional random variable and $\lambda \in \mathbb{R}$. By
constructing a compound Poisson process L_t using

$$(3.1) \qquad\qquad L_t = \sum_{i=1}^{P_t} J_i,$$

where $\{J_i\}_{i=1}^\infty$ denotes a sequence of independent and identically distributed (i.i.d.)
random variables with density function $\phi(z)$, and P_t denotes a Poisson process with
intensity λ that is independent of $\{J_i\}_{i=1}^\infty$, the resulting jump measure $\nu(\mathrm{d}z)$ associated
with L_t precisely corresponds to $\lambda\phi(z)\,\mathrm{d}z$. Accordingly, the jumps can be simulated
on each trajectory based on the procedure illustrated in Figure 3. Initially, a sequence
of exponential distributions $\{E_i\}_{i=1}^{m+1}$ with parameter λ is generated. Subsequently,
the cumulative sum of these random variables yields the arrival times $t_1, t_2, \ldots, t_m$
of the Poisson process P_t. Finally, the density function $\phi(z)$ is sampled m times to
simulate the jumps at each time point. This approach enables the determination of
the jump occurrence times and their corresponding magnitudes.

In the numerical experiments, the neural network architecture is composed of a
$(d+1)$- dimensional input, a linear layer, five residual blocks, a linear layer, and a two-
dimensional output, as illustrated in Figure 1. Each residual block includes two linear
layers mapping from 25-dimensional space to 25-dimensional space and a residual
connection. In high-dimensional experiments, the width of the network is adjusted
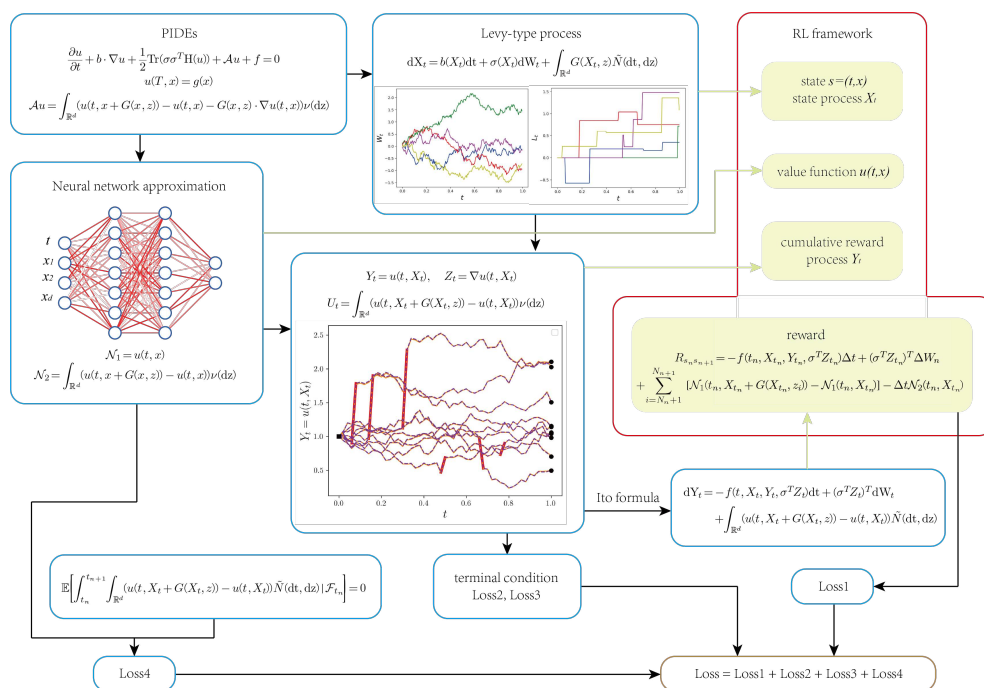
FIG. 2. *The diagram of solving high-dimensional PIDEs with jumps by the TD method. The PIDE under consideration is associated with a set of Lévy-type forward-backward stochastic processes. These processes can be effectively characterized and addressed within the framework of RL, as depicted in the highlighted red box (color available online only) in the figure. By employing neural networks, the calculation of the loss function becomes feasible through the utilization of TD methods. Consequently, the model parameters can be updated accordingly to facilitate the resolution of the PIDE.*
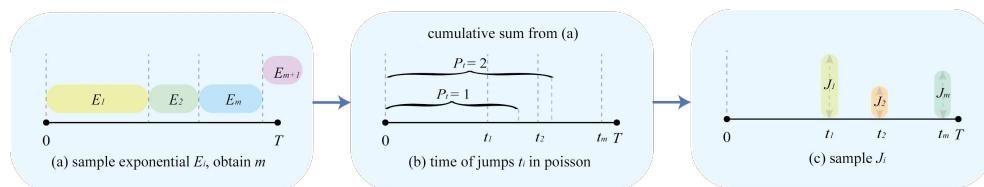

FIG. 3. *Simulation of jumps. To simulate the jumps on each trajectory, the following steps are taken:* (a) *A sequence of exponential distributions with parameter λ is generated.* (b) *The cumulative sum of this sequence yields the arrival times of the Poisson process on the respective trajectory.* (c) *At each arrival time, a single sample is drawn from the distribution $\phi(z)$ to determine the value of the jump at that particular moment.*

proportionally to the dimension. The *tanh* activation function is employed, and the Adam optimizer is chosen with an initial learning rate of 5×10^{-5}. The learning rate is divided by 5 every 5000 parameter updates during the training process. All experiments adopt the same random seed of 2023. Considering the characteristics of BSDEs, the L_1 relative error of Y_0 is employed as the evaluation of the model. The experiments are conducted on a Tesla V100 GPU with 32GB memory and 40GB RAM, utilizing 10 cores.

3.1. One-dimensional pure jump process. One-dimensional Lévy processes are widely encountered in various fields, such as insurance, hedging, and pricing [8]. To better illustrate the abilities of our method in computing the PIDEs, this subsection specifically focuses on the one-dimensional pure jump process to avoid potential challenges arising from higher dimensions and Brownian motion. We aim to solve the following PIDE:

$$(3.2) \qquad \begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \displaystyle\int_{\mathbb{R}} \left(u(t, xe^z) - u(t,x) - x(e^z - 1)\dfrac{\partial u}{\partial x}(t,x) \right) \nu(\mathrm{d}z) = 0, \\ u(T,x) = x, \end{cases}$$

where $\nu(\mathrm{d}z) = \lambda\phi(z)\,\mathrm{d}z$, $\phi(z) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}$, and $x \in \mathbb{R}$. The equation possesses an exact solution of $u(t,x) = x$, and the corresponding Lévy-type processes are

$$(3.3) \qquad \mathrm{d}X_t = \int_{\mathbb{R}} X_t(e^z - 1)\widetilde{N}(\mathrm{d}t, \mathrm{d}z),$$

$$(3.4) \qquad \mathrm{d}Y_t = \int_{\mathbb{R}} (u(t, X_t e^z) - u(t, X_t))\,\widetilde{N}(\mathrm{d}t, \mathrm{d}z).$$

The initial value of the process is set to $X_0 = 1$, and the time interval $[0,1]$ is divided into $N = 50$ equal partitions. The intensity of the Poisson process is $\lambda = 0.3$, with the jumps following a normal distribution with $\mu = 0.4$, $\sigma = 0.25$. A total of $M = 1000$ trajectories are employed. The training consists of 400 iterations, with 50 parameter updates in each iteration, resulting in a cumulative total of 20,000 parameter updates.

Figure 4(a) illustrates the evolution of the relative error of Y_0 throughout the entire training process, while Table 1 presents the specific values of the relative error of Y_0 during the initial and final 5000 steps of training. Figure 4(b) depicts the relative error of the neural network's approximation for Y_t at each time t. Our method achieves a relative error of 0.02% for Y_0. It is noteworthy that the error decreases to approximately 1% after only 1000 parameter updates, and with sufficient training, the error magnitude reaches the order of 10^{-4}. To visually assess the results, Figure 5 illustrates 30 trajectories, among which five exhibit jumps. The results illustrate the
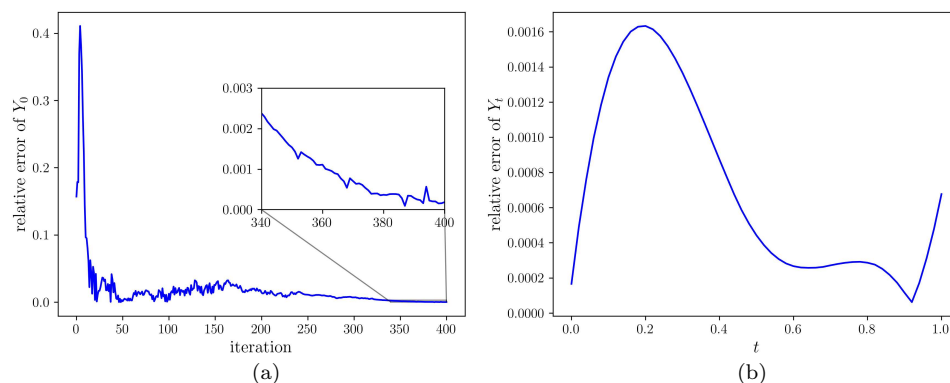

FIG. 4. *Relative error of the one-dimensional pure jump problem.* (a) *The evolution of the relative error of Y_0 with respect to the number of iterations. The exact value is $Y_0 = 1$, and the relative error converges to 0.02%.* (b) *The relative error of the neural network's approximation for Y_t at different time t.*

TABLE 1
Relative error of Y_0 during the initial and final 5000 updates of training in the one-dimensional pure jump problem. The exact value is $Y_0 = 1$.

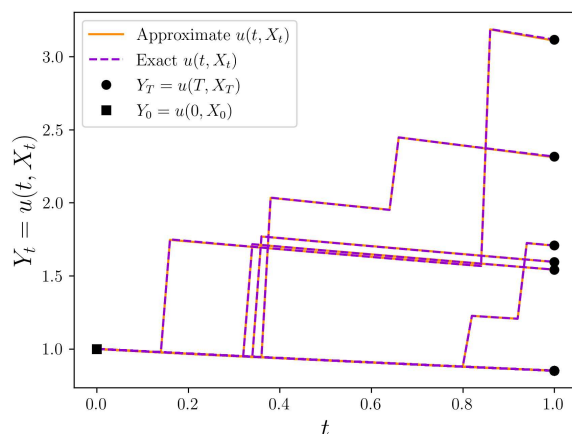| Iteration	Y_0	Relative error	Iteration	Y_0	Relative error
0	1.1570	15.70%	15000	0.9929	0.71%
500	1.0965	9.65%	15500	0.9943	0.57%
1000	1.0033	0.33%	16000	0.9946	0.54%
1500	1.0313	3.13%	16500	0.9968	0.32%
2000	0.9853	1.47%	17000	0.9976	0.24%
2500	1.0013	0.13%	17500	0.9985	0.15%
3000	0.9974	0.26%	18000	0.9989	0.11%
3500	0.9862	1.38%	18500	0.9993	0.07%
4000	0.9896	1.04%	19000	0.9996	0.04%
4500	0.9837	1.63%	19500	0.9997	0.03%
5000	0.9875	1.25%	20000	0.9998	0.02%

FIG. 5. *The visualization of the trajectories for the one-dimensional pure jump problem. 30 trajectories are displayed, and five trajectories exhibit jumps, while the remaining 25 trajectories coincide entirely due to the absence of Brownian motion.*

rapid convergence and high precision achieved by our method in one-dimensional pure jump problems. In summary, our method demonstrates remarkable performance.

3.2. Robustness. The proposed method involves various parameters, which may not always align with the ideal conditions encountered in practical scenarios. The effectiveness of our method has been demonstrated in the one-dimensional pure jump problem. In this subsection, the performance of the method under different parameter settings is evaluated to assess its robustness, including the number of trajectories, the number of time intervals, jump intensity, jump form, and TD step size. The experiment is conducted using the following PIDE:

$$(3.5) \quad \begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \epsilon x \dfrac{\partial u}{\partial x}(t,x) + \dfrac{1}{2}\theta^2 \dfrac{\partial^2 u}{\partial x^2}(t,x) \\ \quad + \displaystyle\int_{\mathbb{R}} \left(u(t,xe^z) - u(t,x) - x(e^z-1)\dfrac{\partial u}{\partial x}(t,x) \right) \nu(\mathrm{d}z) = \epsilon x, \\ u(T,x) = x, \end{cases}$$

where $\nu(\mathrm{d}z) = \lambda\phi(z)\,\mathrm{d}z$, and $x \in \mathbb{R}$. The equation possesses an exact solution of $u(t,x) = x$, and the corresponding Lévy-type processes are

$$(3.6) \qquad \mathrm{d}X_t = \epsilon X_t\,\mathrm{d}t + \theta\,\mathrm{d}W_t + \int_{\mathbb{R}} X_t(e^z - 1)\widetilde{N}(\mathrm{d}t, \mathrm{d}z),$$

$$(3.7) \qquad \mathrm{d}Y_t = \epsilon X_t\,\mathrm{d}t + \theta Z_t\,\mathrm{d}W_t + \int_{\mathbb{R}} \left(u(t, X_t e^z) - u(t, X_t)\right)\widetilde{N}(\mathrm{d}t, \mathrm{d}z),$$

and the initial value is set to $X_0 = 1$.

(a) Trajectories and intervals. In the computation of the loss function, Monte Carlo methods are employed to estimate the expectations. Therefore, the number of trajectories M used can potentially influence the method's performance. The number of time intervals N is associated with the discretization of the stochastic process, as well as the number of optimizations in each iteration. Hence, it is an important factor in assessing the method's robustness. Here we assume that the jump form will follow a normal distribution $\phi(z) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}$ with $\mu = 0.4$, $\sigma = 0.25$ and the intensity of the Poisson process is set to $\lambda = 0.3$, maintaining consistency with subsection 3.1. Additionally, we set $\epsilon = 0.25$, $\theta = 0$ in (3.5) and the termination time $T = 1$.

Since the number of time intervals N affects the number of optimizations performed in each iteration, we report the results for three different numbers of iterations: 250, 500, and 750. For example, in the case of 500 iterations and $N = 40$ time intervals, this implies a total of $500 * 40 = 20,000$ parameter updates. The results are presented in Table 2. The findings reveal that increasing the number of trajectories M can effectively reduce the error at convergence, with a more pronounced improvement observed when the number of trajectories increases from $M = 125$ to $M = 250$. On the other hand, altering the number of time intervals between $N = 20, 40, 80$ does not exert a significant influence on the error at convergence. For trajectory numbers of $M = 250$ or higher, the error at convergence almost consistently remains within the order of 10^{-3}. Therefore, our method does not require a high number of trajectories and is not sensitive to the number of time intervals.

(b) Lévy measure. The form of the Lévy measure plays a decisive role in nonlocal terms. Here we aim to evaluate the performance of our method under different Poisson intensities λ and jump forms $\phi(z)$. A total of $M = 250$ trajectories are employed, and the time interval $[0,1]$ is evenly divided into $N = 50$ intervals. The parameters in (3.5) are set as $\epsilon = 0$, $\theta = 0.4$. The training process comprises 400 iterations, resulting in a total of 20,000 parameter updates. For the conducted tests, the Poisson intensities are systematically varied from 0.3 to 1.8. Additionally, the following four distinct jump distributions $\phi(z)$ are utilized: (1) normal distribution with $\mu = 0.4$, $\sigma = 0.25$, (2) uniform distribution with $\delta = 0.4$, (3) exponential distribution with $\lambda_0 = 3$, and (4) Bernoulli distribution with $a_1 = -0.2$, $a_2 = 0.4$, $p = 0.7$:

TABLE 2

Relative errors of Y_0 for different numbers of trajectories M and time intervals N in the robustness checks. Parameters are updated N times in each iteration.

Iterations	250			500			750		
Intervals N	20	40	80	20	40	80	20	40	80
$M = 1000$	7.687%	1.671%	1.723%	0.303%	0.660%	0.545%	0.276%	0.159%	0.377%
$M = 500$	2.988%	0.412%	2.611%	0.245%	1.385%	0.665%	0.219%	0.402%	0.639%
$M = 250$	0.972%	1.671%	2.847%	0.264%	0.598%	0.502%	2.122%	0.139%	0.364%
$M = 125$	4.100%	5.687%	4.303%	3.272%	0.662%	1.968%	1.700%	0.489%	1.691%

TABLE 3

Relative errors of Y_0 under different Poisson intensities λ and jump forms $\phi(z)$ in the robustness checks. The specific forms of the four distributions are given in (3.8). The experiments were all conducted with $M = 250$ trajectories, $N = 50$ time intervals, and a total of 20,000 parameter updates.

Y_0 relative error	Normal	Exponential	Uniform	Bernoulli
$\lambda = 0.3$	0.051%	0.475%	0.376%	0.752%
$\lambda = 0.6$	0.117%	0.393%	2.450%	2.469%
$\lambda = 0.9$	0.340%	0.910%	1.355%	1.997%
$\lambda = 1.2$	1.111%	0.567%	1.048%	1.202%
$\lambda = 1.5$	1.960%	1.591%	1.202%	1.227%
$\lambda = 1.8$	1.173%	0.461%	0.167%	1.257%

$$(3.8) \quad \phi(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}, \quad \phi(z) = \begin{cases} \frac{1}{2\delta}, & -\delta \leq z \leq \delta, \\ 0 & \text{else}, \end{cases}$$

$$\phi(z) = \begin{cases} \lambda_0 e^{-\lambda_0 z}, & z \geq 0, \\ 0, & z < 0, \end{cases} \quad \phi(z) = \begin{cases} p, & z = a_1, \\ 1-p, & z = a_2. \end{cases}$$

Table 3 illustrates the relative errors of Y_0 under various Poisson intensities λ and jump forms $\phi(z)$. With the increase of Poisson intensity from $\lambda = 0.3$ to $\lambda = 1.8$, the occurrence frequency of jumps within the $[0, T]$ time interval gradually increases. The results reveal a growing trend in the error for the normal distribution, while the errors for the remaining three distributions show no significant changes. Across almost all scenarios, the relative errors of Y_0 remain within 2%. Figure 6 visually showcases 10 trajectories for each of the four jump forms when the Poisson intensity is $\lambda = 0.3$. Notably, the occurrence of jumps is indicated by bold red lines. The figure illustrates the excellent performance of our method under the four distinct jump forms.

(c) n-step TD. According to the loss function, the previous experiments can be regarded as one-step TD methods. However, an intuitive approach is to investigate the influence of selecting an appropriate TD step n on the method's performance. In this regard, we set $\epsilon = 0.25$, $\theta = 0.4$ in (3.5) for training purposes. A total of $M = 250$ trajectories were employed, and the Poisson process intensity was set to $\lambda = 0.3$, with the jump form being $\phi(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}$ and $\mu = 0.4$, $\sigma = 0.25$. For the sake of computational convenience, it was desirable to have the number of intervals N precisely divisible by the TD step n. Consequently, the time $[0, 1]$ was divided into $N = 60$ intervals, and we examined the TD step $n = 1, 2, 3, 4, 5, 6$.

Due to the reduced number of parameter updates in each iteration with larger step sizes n, and considering both the number of parameter updates and convergence, we performed $M = 500$ iterations for $n = 1, 2$, $M = 1000$ iterations for $n = 3, 4$, and $M = 1500$ iterations for $n = 5, 6$. The results of the relative error of Y_0 for different step sizes n are presented in Table 4. It can be observed that, in this case, the one-step TD method demonstrates the most favorable computational performance. The error does not exhibit a significant trend as the step size increases, and it remains almost consistently below 1%.

3.3. High-dimensional problems. When faced with higher-dimensional problems, many methods encounter a bottleneck due to the exponential increase in computational cost. This phenomenon, known as the curse of dimensionality, poses a significant challenge. In this subsection, our focus shifts towards investigating the
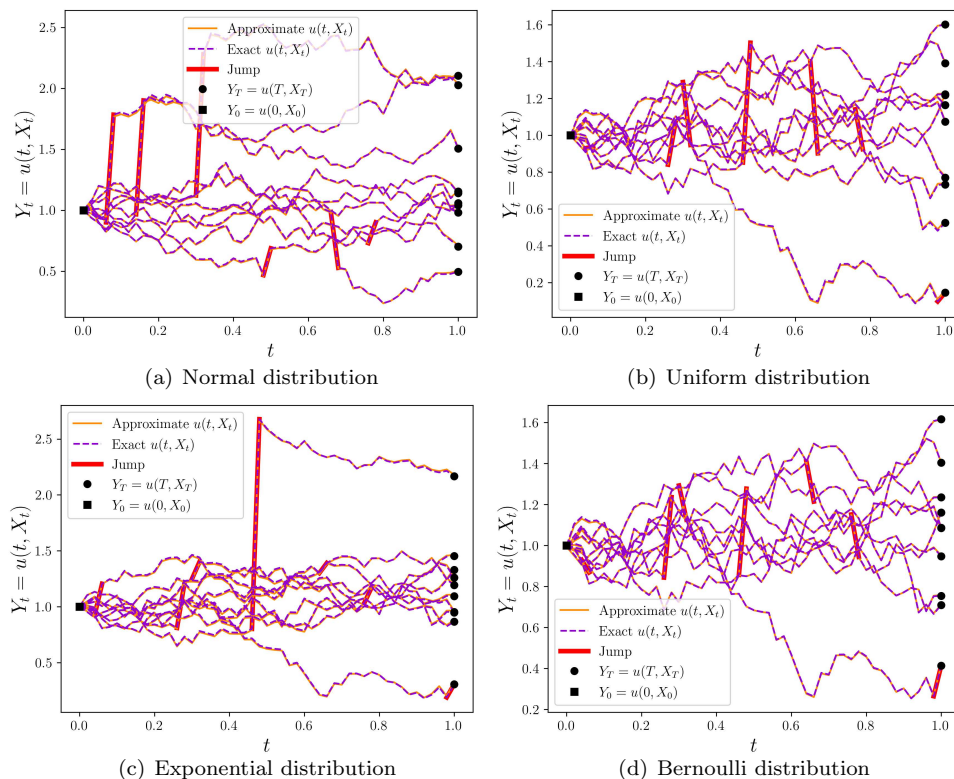
(a) Normal distribution

(b) Uniform distribution

(c) Exponential distribution

(d) Bernoulli distribution

FIG. 6. *The visualization of the trajectories under different jump forms $\phi(z)$ in the robustness checks. To distinguish the jumps with smaller amplitudes from the Brownian motion, the occurrence of jumps is marked with bold red lines (color available online only).*

TABLE 4
Relative errors of Y_0 for different TD step n in the robustness checks. We performed $M = 500$ iterations for $n = 1, 2$, $M = 1000$ iterations for $n = 3, 4$, and $M = 1500$ iterations for $n = 5, 6$.

TD step n	1	2	3	4	5	6
Y_0	0.9998	0.9926	1.0114	1.0088	1.0078	0.9965
relative error of Y_0	0.022%	0.745%	1.137%	0.880%	0.780%	0.354%

performance in high-dimensional scenarios. The experiments are conducted using the following PIDE:

$$(3.9) \quad \begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \dfrac{\epsilon}{2} x \cdot \nabla u(t,x) + \dfrac{1}{2}\mathrm{Tr}(\theta^2 \mathrm{H}(u)) \\ + \displaystyle\int_{\mathbb{R}^d} (u(t,x+z) - u(t,x) - z \cdot \nabla u(t,x))\,\nu(\mathrm{d}z) = \lambda(\mu^2 + \sigma_0^2) + \theta^2 + \dfrac{\epsilon}{d}\|x\|^2, \\ u(T,x) = \dfrac{1}{d}\|x\|^2, \end{cases}$$

with $\epsilon = 0$, $\theta = 0.3$, $x \in \mathbb{R}^d$ and Poisson intensity $\lambda = 0.3$. To highlight the effectiveness of our method in handling high-dimensional problems, we chose the jump form to be

independent multidimensional normal distributions, with a mean of $\mu*(1,1,\ldots,1)$ and a covariance matrix $\sigma^2 I_d$. Here $\mu = 0.1$, $\sigma_0^2 = 0.0001$, and the intention is to maintain uniformity in jump sizes to the greatest extent possible. The equation possesses an exact solution of $u(t,x) = \frac{1}{d}\|x\|^2$. The training process utilizes $M = 500$ trajectories, and the time $[0,1]$ is divided into $N = 50$ intervals. Furthermore, the number of neurons in the linear layers of each block is adjusted to $d + 10$, instead of 25, to accommodate the varying input dimensions d.

In the 100-dimensional experiment, the relative error of Y_0 is found to be 0.548%, with a computational time of approximately 10 minutes. Figure 7(a) depicts the evolution of the relative error of Y_0 throughout the entire training process in the 100-dimensional experiment. Since the model did not converge after 400 iterations, i.e., 20,000 parameter updates, the training was extended to 600 iterations, i.e., 30,000 parameter updates. Furthermore, Figure 7(b) provides a visual representation of five trajectories in the 100-dimensional problem, highlighting the locations of jumps through bold red lines. It is evident that our proposed method achieves a high-precision approximation of the solution in the 100-dimensional problem, while maintaining a computationally feasible runtime.

Furthermore, the performance of our method was evaluated in various dimensions d. The experimental setup remained consistent with the 100-dimensional experiment. Table 5 presents the relative error of Y_0 and the program's execution time across different dimensions d. It is noteworthy that regardless of low or high dimensions, the relative error of Y_0 generally remains within the range of 10^{-3}. All experiments were conducted using $M = 500$ trajectories, and it is evident that our method exhibits a minimal sensitivity to the number of trajectories in high-dimensional problems. As the dimensionality increases, the program's execution time exhibits a roughly linear growth, ranging from 5 to 10 minutes, which is entirely acceptable. In summary, our method demonstrates remarkable performance across different dimensions and shows no significant curse of dimensionality.

Finally, we slightly modified (3.9) to consider the following high-dimensional PIDE with the coupled drift term and the coupled diffusion term
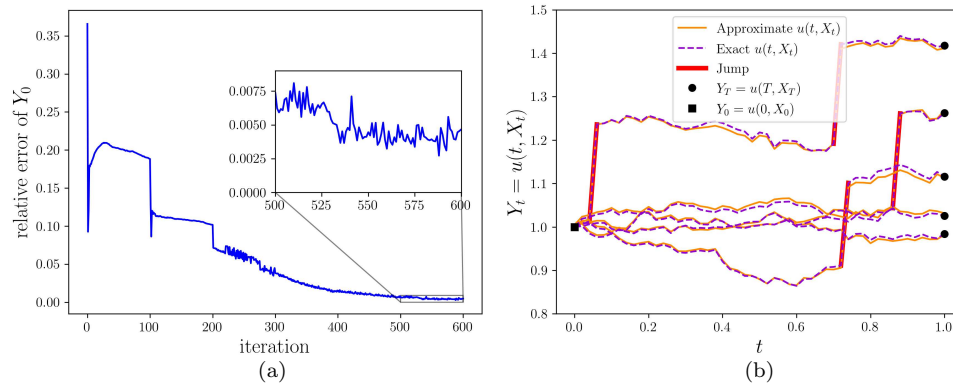

(a)   (b)

FIG. 7. *Result of the 100-dimensional problem. The exact value of Y_0 is 1, and the relative error of the neural network approximation is 0.548%. (a) The evolution of the relative error of Y_0 with the number of iterations. (b) Visualization of five trajectories under the neural network's approximation and the exact solution.*

TABLE 5

The relative error of Y_0 and runtime for different dimensions d. The experiments were conducted on a Tesla V100 GPU with 32GB memory and 40GB RAM.

Dimension	2	4	6	8	10
Y_0 relative error	0.954%	0.251%	0.025%	0.671%	1.895%
Time (s)	332	336	338	356	364
Dimension	20	30	40	50	60
Y_0 relative error	0.702%	1.221%	0.956%	0.219%	0.944%
Times (s)	368	396	401	423	506
Dimension	70	80	90	100	-
Y_0 relative error	0.044%	0.277%	0.460%	0.548%	-
Time (s)	529	565	584	638	-

(3.10)
$$
\begin{cases}
\dfrac{\partial u}{\partial t}(t,x) + \dfrac{\epsilon}{2}\|x\|\, x \cdot \nabla u(t,x) + \dfrac{1}{2}\mathrm{Tr}(\sigma\sigma^T \mathrm{H}(u)) \\[2mm]
+ \displaystyle\int_{\mathbb{R}^d} \left(u(t,x+z) - u(t,x) - z\cdot\nabla u(t,x)\right)\nu(\mathrm{d}z) = \lambda(\mu^2 + \sigma_0^2) + \dfrac{2d-2}{d}\theta^2 + \dfrac{\epsilon}{d}\|x\|^3, \\[2mm]
u(T,x) = \dfrac{1}{d}\|x\|^2.
\end{cases}
$$

Here $\epsilon = 0.05$, $\theta = 0.2$, Poisson intensity $\lambda = 0.3$, and the diffusion term is given by

$$
\sigma = \theta
\begin{pmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
1 & 1 & 0 & 0 & \cdots & 0 \\
0 & 1 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & 0 \\
0 & 0 & 0 & \cdots & 1 & 1
\end{pmatrix}.
$$

The exact solution is $u(t,x) = \frac{1}{d}\|x\|^2$. In this numerical test, we use the same setups as in the previous example, including the selection of the jump form, the discretization of the time interval, the number of trajectories, and the number of iterations. We test the cases for dimensions $d = 25$, 50, 75, and 100 and visualize their trajectories in Figure 8. The relative errors of Y_0 are 0.743% for $d = 25$, 1.910% for $d = 50$, 2.412% for $d = 75$, and 2.387% for $d = 100$. It is seen that our method still performs well with the existence of the coupled drift term and coupled diffusion term.

4. Conclusion. In this work, a framework based on RL is proposed to solve high-dimensional PIDEs. A set of Lévy-type stochastic processes is introduced to characterize the solutions of the PIDE, and an RL model is built upon these processes. Deep neural networks are employed to represent the solutions of the original PIDE and nonlocal terms, enabling a step-by-step simulation of the system. TD learning is utilized, and its error is employed as the loss function, allowing parameter updates without the need to wait for completing the entire trajectory. Additionally, termination conditions and the properties of nonlocal terms are incorporated into the loss function. The entire training process can be completed with a relatively low computational cost.

The numerical experiments demonstrate the efficacy of the proposed method in solving 100-dimensional problems with a relative error on the order of 10^{-3} and one-dimensional pure jump problems with an relative error on the order of 10^{-4}. The
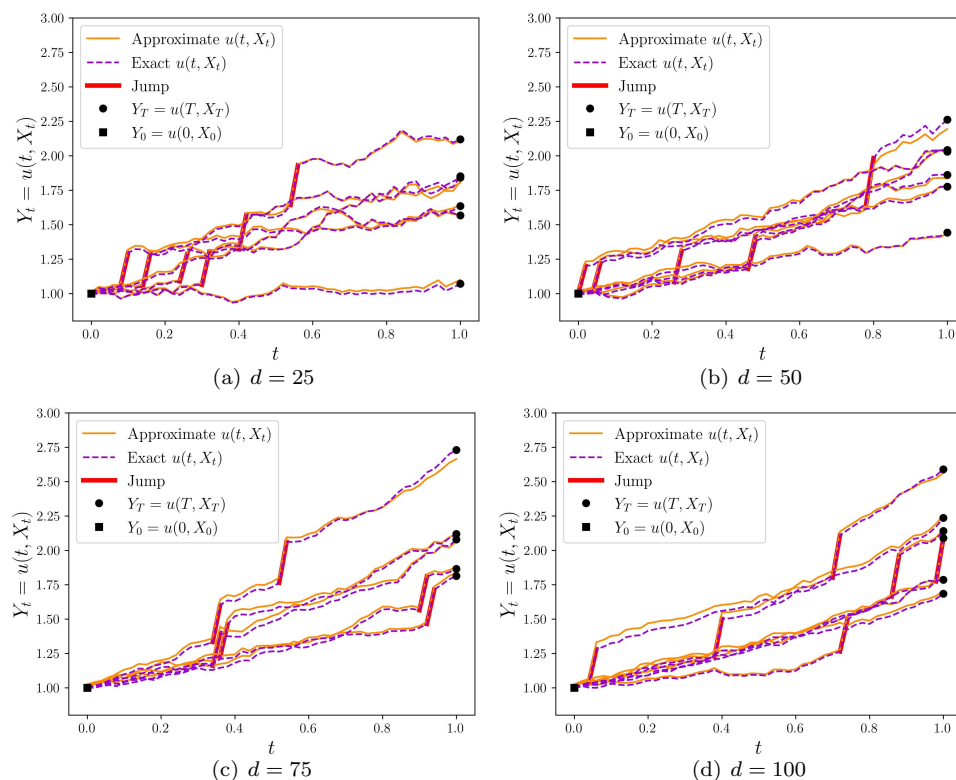
Fig. 8. *The visualization of the trajectories of high-dimensional problems with the coupled drift term and the couple diffusion term.*

computational cost of the method exhibits approximately linear growth with increasing dimensionality, while maintaining a low requirement for the number of trajectories in high-dimensional cases. Robustness tests reveal favorable outcomes for different forms and intensities of jumps, without the need for dense time interval divisions or large TD step sizes. These findings indicate the broader applicability of the proposed method. The error analysis is left for future work. Furthermore, the extension of this method to more general problems, such as inverse problems involving Lévy processes with available data, warrants further investigation in future work.

REFERENCES

[1] F. ABERGEL AND R. TACHET, *A nonlinear partial integro-differential equation from mathematical finance*, Discrete Contin. Dyn. Syst. Ser. A, 27 (2010), pp. 907–917.

[2] D. APPLEBAUM, *Lévy Processes and Stochastic Calculus*, Cambridge University Press, 2009.

[3] S. CAI, Z. MAO, Z. WANG, M. YIN, AND G. E. KARNIADAKIS, *Physics-informed neural networks (PINNs) for fluid mechanics: A review*, Acta Mech. Sin., 37 (2021), pp. 1727–1738.

[4] J. CASTRO, *Deep learning schemes for parabolic nonlocal integro-differential equations*, Partial Differ. Equ. Appl., 3 (2022), 77.

[5] P.-H. CHIU, J. C. WONG, C. OOI, M. H. DAO, AND Y.-S. ONG, *CAN-PINN: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method*, Comput. Methods Appl. Mech. Engrg., 395 (2022), 114909.

[6] R. Cont and E. Voltchkova, *A finite difference scheme for option pricing in jump diffusion and exponential Lévy models*, SIAM J. Numer. Anal., 43 (2005), pp. 1596–1626, https://doi.org/10.1137/S0036142903436186.

[7] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, *Scientific machine learning through physics–informed neural networks: Where we are and what's next*, J. Sci. Comput., 92 (2022), 88.

[8] L. Delong, *Backward Stochastic Differential Equations with Jumps and Their Actuarial and Financial Applications*, Springer, 2013.

[9] J. Duan, *An Introduction to Stochastic Dynamics*, Cambridge Texts Appl. Math. 51, Cambridge University Press, 2015.

[10] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12.

[11] R. Frey and V. Köck, *Deep neural network algorithms for parabolic PIDEs and applications in insurance mathematics*, in Mathematical and Statistical Methods for Actuarial Sciences and Finance: MAF 2022, Springer, 2022, pp. 272–277.

[12] A. Gnoatto, M. Patacca, and A. Picarelli, *A Deep Solver for BSDEs with Jumps*, preprint, arXiv:2211.04349, 2022.

[13] A. Goswami, J. Patel, and P. Shevgaonkar, *A system of non-local parabolic PDE and application to option pricing*, Stoch. Anal. Appl., 34 (2016), pp. 893–905.

[14] D. Goswami, A. K. Pani, and S. Yadav, *Optimal error estimates of two mixed finite element methods for parabolic integro-differential equations with nonsmooth initial data*, J. Sci. Comput., 56 (2013), pp. 131–164.

[15] 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), pp. 349–380.

[16] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510.

[17] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, *Physics-informed machine learning*, Nat. Rev. Phys., 3 (2021), pp. 422–440.

[18] N. I. Kavallaris and T. Suzuki, *Non-local Partial Differential Equations for Engineering and Biology*, Math. Model. Anal. 31, Springer, 2018.

[19] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, *hp-VPINNs: Variational physics-informed neural networks with domain decomposition*, Comput. Methods Appl. Mech. Engrg., 374 (2021), 113547.

[20] M. Köppen, *The curse of dimensionality*, in Proceedings of the 5th Online World Conference on Soft Computing in Industrial Applications (WSC5), Vol. 1, 2000, pp. 4–8.

[21] Y. Kwon and Y. Lee, *A second-order finite difference method for option pricing under jump-diffusion models*, SIAM J. Numer. Anal., 49 (2011), pp. 2598–2617, https://doi.org/10.1137/090777529.

[22] W. Li, C. Zhang, C. Wang, H. Guan, and D. Tao, *Revisiting PINNs: Generative Adversarial Physics-Informed Neural Networks and Point-Weighting Method*, preprint, arXiv:2205.08754, 2022.

[23] S. Liang and H. Yang, *Finite Expression Method for Solving High-Dimensional Partial Differential Equations*, preprint, arXiv:2206.10121, 2022.

[24] Y. Liao and P. Ming, *Deep Nitsche method: Deep Ritz method with essential boundary conditions*, Commun. Comput. Phys., 29 (2021), pp. 1365–1384.

[25] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, SIAM Rev., 63 (2021), pp. 208–228, https://doi.org/10.1137/19M1274067.

[26] L. Lyu, Z. Zhang, M. Chen, and J. Chen, *MIM: A deep mixed residual method for solving high-order partial differential equations*, J. Comput. Phys., 452 (2022), 110930.

[27] L. D. McClenny and U. M. Braga-Neto, *Self-adaptive physics-informed neural networks*, J. Comput. Phys., 474 (2023), 111722.

[28] B. Moseley, A. Markham, and T. Nissen-Meyer, *Finite basis physics-informed neural networks (FBPINNs): A scalable domain decomposition approach for solving differential equations*, Adv. Comput. Math., 49 (2023), 62.

[29] O. Pablo and D. Ciro, *A finite elements approach for spread contract valuation via associated two-dimensional PIDE*, Comput. Appl. Math., 42 (2023), 15.

[30] M. Raissi, *Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations*, in Peter Carr Gedenkschrift: Research Advances in Mathematical Finance, 2024, pp. 637–655.

[31] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.

[32] J. SIRIGNANO AND K. SPILIOPOULOS, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364.

[33] W. WANG, J. WANG, J. LI, F. GAO, AND Y. FU, *Deep Learning Numerical Methods for High-Dimensional Fully Nonlinear PIDEs and Coupled FBSDEs with Jumps*, preprint, arXiv:2301.12895, 2023.

[34] C. L. WIGHT AND J. ZHAO, *Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks*, Commun. Comput. Phys., 29 (2021), pp. 930–954.

[35] J. YU, L. LU, X. MENG, AND G. E. KARNIADAKIS, *Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems*, Comput. Methods Appl. Mech. Engrg., 393 (2022), 114823.

[36] Y. ZANG, G. BAO, X. YE, AND H. ZHOU, *Weak adversarial networks for high-dimensional partial differential equations*, J. Comput. Phys., 411 (2020), 109409.

[37] S. ZENG, Y. CAI, AND Q. ZOU, *Deep neural networks based temporal-difference methods for high-dimensional parabolic partial differential equations*, J. Comput. Phys., 468 (2022), 111503.