Punjab University Journal of Mathematics (2025), 57(02), 165-179
https://doi.org/10.52280/pujm.2025.57(02)03

Innovative Neural Network Approach for Solving TDKS Equations with Jensen's Inequality

*Adeel Farooq
Department of Mathematics,
COMSATS University Islamabad, Lahore Campus

Muhammad Tayyab Chaudhary
Department of Computer Science,
COMSATS University Islamabad, Lahore Campus

Muhammad Yousaf
Department of Mathematics,
COMSATS University Islamabad, Lahore Campus

Samra Sarwar
Department of Mathematics,
COMSATS University Islamabad, Lahore Campus

Farooq Ahmad
Department of Computer Science,
COMSATS University Islamabad, Lahore Campus

*Corresponding Author
adeelfarooq@cuilahore.edu.pk

Received: 10 Feb, 2025 / Accepted: 18 July, 2025 / Published online: 20 August, 2025

Abstract. We propose a novel neural network-based approach for solv-ing the Time-Dependent Kohn-Sham (TDKS) equations, central to Time-Dependent Density Functional Theory (TDDFT). Focusing on the one-electron case, where the TDKS reduces to the time-dependent Schrödinger equation, we employ modified Physics-Informed Neural Networks (PINNs) incorporating Jensen's inequality in place of the traditional Mean Squared Error (MSE) loss. This convex formulation improves training stability and solution accuracy. Compared to the classical Runge-Kutta (RK4) method, our approach achieves comparable accuracy while demonstrating supe-rior scalability and smoother convergence, especially for stiff or nonlinear dynamics. This work establishes a foundation for extending neural PDE solvers to more complex quantum systems.that the classroom teaching ex-periment began.

AMS (MOS) Subject Classification Codes: 92B20, 68T07
Key Words: PINNS . Jensen's inequality . Mean squared error . Runge-Kutta methods . Kohn-sham equations.

1. Introduction

The Time-Dependent Kohn-Sham (TDKS) [7] equations play a fundamental role in the study of quantum systems [8] and form the basis of Time-Dependent Density Functional Theory (TDDFT), which is widely used in electronic structure calculations. These equations describe the quantum dynamics of interacting electrons through a set of non-interacting single-particle equations [15]. Due to their ability to capture dynamic responses of molecular and extended systems [9], they are employed in various fields such as computational chemistry, solid-state physics, and materials science. However, solving these

equations accurately and efficiently remains a challenging task, especially for large systems or when the dynamics are nonlinear and stiff.

The traditional way to solve such equations has been the use of numerical time-stepping methods like the Runge-Kutta method of order 4 (RK4) [3, 18]. Although RK4 is quite stable and accurate in certain settings, it becomes computationally expensive and often impractical when dealing with high-dimensional or stiff systems. Moreover, the stepwise nature of RK4 can introduce numerical instability and accumulated errors over long time spans or for oscillatory quantum behaviours. Hence, there arises a strong motivation to explore alternative techniques that can handle such limitations and provide reliable results [12, 16].

Recently, machine learning-based techniques [11, 14], particularly Physics-Informed Neural Networks (PINNs) [13], have shown promising potential in solving partial differential equations. PINNs offer an advantage by embedding physical laws directly into the loss function of the neural network, which allows the model to learn the solution of differential equations from data and constraints simultaneously. This technique not only avoids traditional time-stepping but also provides a flexible and scalable framework for high-dimensional systems. Despite this, the commonly used loss function in PINNs is the Mean Squared Error (MSE), which does not always yield the most stable or accurate learning behaviour, particularly in the presence of nonlinearities.

Parallel to this, recent advancements in unsupervised deep learning have expanded the applicability of neural network methods across a range of complex dynamical systems. For instance, Mukheimer and Ejaz [10] applied neural networks to model trophic chains involving cannibalism and harvesting, demonstrating strong agreement between unsupervised models and classical dynamical behaviour. Similarly, Abodayeh and Nawaz [1] compared a third-order numerical scheme with neural network-based solvers for SEIR epidemic modeling, indicating the reliability of neural networks in stiff system dynamics. In another application, Arif et al. [2] used explainable machine learning for chronic kidney disease prediction, illustrating how unsupervised or semi-supervised learning frameworks can adapt to real-world nonlinearities. These examples collectively highlight the broader utility of unsupervised and physics-informed neural networks across disciplines, reinforcing the relevance of extending such methods to quantum mechanical systems.

This paper proposes a novel approach by modifying the PINNs framework through the use of Jensen?s inequality in the computation of the physics-based loss component. Jensen?s inequality [4, 5], due to its mathematical properties related to convex functions, helps in controlling the behaviour of error functions and can improve the convergence of neural networks dealing with physical models. To the best of our understanding, this combination of PINNs with Jensen?s inequality has not been previously applied in the context of solving TDKS equations [12].

Therefore, the main aim of this research is to evaluate the performance of this modified PINN framework, compare it against the traditional RK4 method and the standard PINN

model using MSE, and highlight the advantages of using Jensen?s inequality [3]. This paper also explores the accuracy, computational efficiency, and scalability of the proposed method. The novelty of our contribution lies in this unique integration of mathematical inequality into a machine learning framework for quantum system modelling [9, 13, 16, 18], which could provide new directions for solving advanced problems in quantum physics.
Despite the widespread use of traditional solvers such as Runge-Kutta (RK4) for quantum systems, these methods often struggle with scalability, stiffness, and long-term accuracy in nonlinear or high-dimensional problems. Recent advances in neural network-based solvers like PINNs offer a promising direction, yet their effectiveness is often limited by the choice of loss function?typically Mean Squared Error (MSE)?which may not sufficiently control optimization dynamics, especially in stiff equations. Motivated by this gap, we explore a novel integration of Jensen?s inequality into the PINN framework, aiming to improve training stability and solution accuracy. This approach leverages the convex structure of Jensen's inequality to impose a tighter bound on the loss function, which we hypothesize can enhance convergence and generalization. Our work is thus driven by the need to develop more robust, accurate, and mathematically grounded neural solvers for quantum dynamical systems like the TDKS equations.
The rest of the paper is arranged as: Section 2 describes the methods applied to solve TDSK equations. Section 3 describes the proposed architecture. Section 4 describes the results and Section 5 presents the conclusion.

2. METHODS

This section outlines the methods employed to solve the Time-Dependent Kohn-Sham (TDKS) equations: the Runge-Kutta 4th Order (RK4) method, Physics-Informed Neural Networks (PINNs), and the application of Jensen?s inequality for loss computation. These methods exhibit varying strengths in capturing the complex dynamics of the TDKS system. The RK4 method is a well-established numerical technique for solving ordinary and partial differential equations by approximating derivatives at multiple points within a given time step. In this study, we utilized the SciPy and NumPy libraries [11, 17] to implement efficient time-stepping of the wavefunction. The TDKS equations were solved by propagating the wavefunction using time-evolution operators derived via RK4 and PINNs, with Jensen?s inequality serving as the foundation for data loss computation.
We give here few details about this inequality below.
Jensen?s inequality describes a fundamental property of convex and concave functions concerning expected values. If f is a convex function, X is a random variable, and $E[X]$ denotes the expected value of X, then:

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)],$$

as stated in [4]. This implies that the function value at the mean of X is less than or equal to the mean of the function values applied to X, provided f is convex. Conversely, if f is a concave function, the inequality reverses:

$$f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)].$$

In neural networks, especially probabilistic models like Variational Autoencoders (VAEs), Jensen?s inequality is used to derive the Evidence Lower Bound (ELBO) in the training of

models with probabilistic layers. It ensures that the approximation error in such models is bounded [13].
The Adam optimizer plays a crucial role in Physics-Informed Neural Networks (PINNs) by efficiently adjusting the network's weights and biases during training to minimize the loss function. Its role is particularly significant in the context of PINNs due to the unique challenges posed by these networks. For the sake of completeness, we give here the algorithm for Adam's Optimizer [6].
Input: Learning rate α, β_1, β_2, ϵ, initial parameters θ_0, loss function $L(\theta)$

Algorithm 1 Adam Optimizer

Initialize $m_0 = 0$, $v_0 = 0$, $t = 0$
while not converged **do**
$t \leftarrow t + 1$
Compute gradient: $g_t = \nabla L(\theta_{t-1})$
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (First moment estimate)
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (Second moment estimate)
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (Bias correction)
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (Bias correction)
$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while

Now, we discuss PINNs, which lies at the core of our proposed architecture.
PINNs are a class of deep learning models that integrate physical laws, typically expressed as partial differential equations (PDEs), into the training process of neural networks. Unlike conventional machine learning models that rely solely on labeled data, PINNs leverage physics-based constraints to guide the training process, making them particularly suitable for solving forward and inverse problems in scientific and engineering domains.

3. Architecture of PINNs

The core architecture of a PINNs resembles a standard feedforward neural network, consisting of:
Input Layer: Encodes spatial and temporal coordinates (e.g., x, y, t).
Hidden Layers: Learn representations of the solution.
Output Layer: Predicts the solution to the problem, such as scalar fields (e.g., temperature, pressure) or vector fields (e.g., velocity).
Thus PINNs is a machine learning method that utilizes neural networks to solve differential equations, while enforcing residuals corresponding to the differential equation and the contstraints through the loss function itself. We transform the TDKS equations to a residual form and train the neural network on this residual:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{\partial \psi(t_i)}{\partial t} - H(t_i)\psi(t_i) \right|^2$$

Where N represents the number of training points and $H(t_i)$ is the Hamiltonian at each time step t_i.
Since PINNs provides a high level of flexibility when solving problems with complicated, and can learn from data directly, they seem to be an ideal method for the solution of high-dimensional and non-linear TDKS equations.
One of the most frequently used approaches to model time-dependent electronic dynamics is TDDFT:

$$i\frac{\partial\phi_j(r,t)}{\partial t} = \left(-\frac{1}{2}\nabla^2 + v_s[n](r,t)\right)\phi_j(r,t),$$

where $\phi_j(r,t)$ is the wavefunction of the j-th Kohn-Sham orbital, $v_s[n](r,t)$ is the Kohn-Sham potential, and $n(r,t)$ is the electronic density.
The complex wavefunction :

$$\phi_j(r,t) = u_j(r,t) + iv_j(r,t),$$

leads to the following coupled PDEs:

$$\frac{\partial u_j}{\partial t} = -\frac{1}{2}\nabla^2 v_j + v_s[n]v_j,$$
$$\frac{\partial v_j}{\partial t} = \frac{1}{2}\nabla^2 u_j - v_s[n]u_j.$$

The loss function of the PINN is composed of three parts: physics loss, boundary condition loss, and initial condition loss:

$$\mathcal{L} = \lambda_{\text{physics}}\mathcal{L}_{\text{physics}} + \lambda_{\text{boundary}}\mathcal{L}_{\text{boundary}} + \lambda_{\text{initial}}\mathcal{L}_{\text{initial}}$$

where $\lambda_{\text{physics}}, \lambda_{\text{boundary}}, \lambda_{\text{initial}}$ are regularization parameters. The physics loss ensures the satisfaction of the TDKS equations:

$$\mathcal{L}_{\text{physics}} = \frac{1}{N}\sum_{i=1}^{N}\left(\left|\frac{\partial u_j}{\partial t} + \frac{1}{2}\nabla^2 v_j - v_s[n]v_j\right|^2 + \left|\frac{\partial v_j}{\partial t} - \frac{1}{2}\nabla^2 u_j + v_s[n]u_j\right|^2\right).$$

The boundary condition loss is given by:

$$\mathcal{L}_{\text{boundary}} = \frac{1}{N_{\partial\Omega}}\sum_{i=1}^{N_{\partial\Omega}}\left|u_j(r_i,t) - u_j^{\text{BC}}(r_i,t)\right|^2 + \left|v_j(r_i,t) - v_j^{\text{BC}}(r_i,t)\right|^2.$$

The first part is the initial condition loss that assigns correctly consisting start values for your wavefunction:

$$\mathcal{L}_{\text{initial}} = \frac{1}{N_0}\sum_{i=1}^{N_0}\left|u_j(r_i,0) - u_j^0(r_i)\right|^2 + \left|v_j(r_i,0) - v_j^0(r_i)\right|^2.$$

4. Proposed Architecture

The neural network is trained to minimize the loss function L using the Adam optimizer[6], which is a popular improvement over standard gradient descent. Adam works by adjusting learning rates on its own, based on how the gradients (the values showing the direction to adjust the network's parameters) behave. Specifically, Adam uses two main measures: the average direction of the gradients (called the "first moment") and how much the gradients

vary (called the "second moment"). This helps the network learn faster and more reliably, especially in complex problems.

In the case of PINNs, using the Adam optimizer helps a lot in the following way: PINNs are designed to solve equations by making the network obey physical laws during training. When using Adam, the optimizer can quickly adapt to different parts of the problem, helping the network focus on solutions that align with these laws more closely. For example, while trying to solve the TDKS equations, PINNs with Adam can handle the complicated quantum calculations more efficiently, making the training faster and more stable. This approach, using PINNs and Adam, is useful for simulating large quantum systems since it saves time and handles large data smoothly.

We have incorporated two Loss functions for PINNs: MSE Loss and Jensen's inequlity Loss. The details are given below.

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left(R_u(x_i,t_i)^2 + R_v(x_i,t_i)^2\right),$$

where R_u and R_v are the residuals: and

$$R_u = \frac{\partial u_j}{\partial t} + \frac{1}{2}\nabla^2 v_j - v_s[n]v_j,$$
$$R_v = \frac{\partial v_j}{\partial t} - \frac{1}{2}\nabla^2 u_j + v_s[n]u_j \quad [9].$$

The Jensen's Inequality Loss is given below:

$$\mathcal{L}_{\text{Jensen}} = \left(\frac{1}{N}\sum_{i=1}^{N}(R_u + R_v)\right)^2 \quad [18].$$

While calculating the loss during the training epochs, the loss is sum of two independent values called Physics loss and data loss respectively. The former value is quantified by the application of Jensen's inequality for both imaginary and real parts whereby the aggregated value of Jensen's inequality is considered as the value of the Physics loss for each training epoch. The latter value is calculated using MSE both for imaginary and real parts and the aggregated value is considered as the data loss during each training epoch.

We obtain a base learning rate before the training phase by using the Adam optimizer. This is further exlpained in the later section 3.1.

The layered Architechture of the proposed neural network is shown in the table below explaining the multi layered design consisting of four hidden layers with [200, 100, 100, 100] neurons. We have used Leaky ReLu with suitable alpha value ranging in $[0.1, 0.5]$. The activation function Tanh has been used to preserve the negative values of the wave function. The output has been generated from two output layers one for each real and imaginary part.

There are a total of $41,102$ traininable parameters in the proposed model.

This following algorithm describes the implementation of PINNs for solving the TDSK equation, constructing a neural network with LeakyReLU activations, a fixed learning rate of 0.1 for all layers, and a combination of data loss (via Mean Squared Error) and physics loss (via Jensen's inequality).

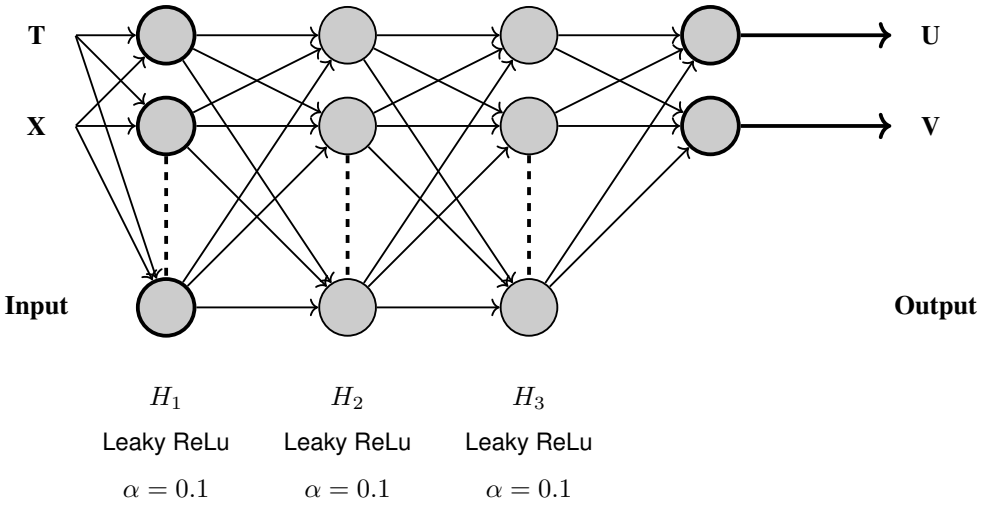


FIGURE 1. Architecture of the Neural Network

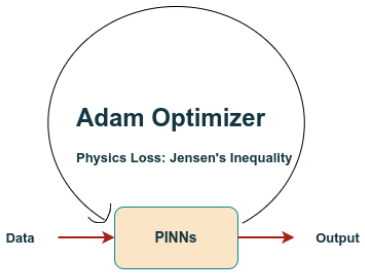


FIGURE 2. Training

The model summary is given in the following figure.

4.1. **Learning rate optimization.** Learning rate is used as an input to Adam optimizer. An optimum learning rate value ensures convergence of the training process in finite epochs. We have achieved this by testing a range of learning rates within a finite range of 10^{-3} to 1 for this implementaion as shown in the figure below. The corresponding loss value against each value of learning rate is plotted and the most stable learning rate is between 10^{-3} to 10^{-2}. We chose the former value 10^{-3} as the value of the learning rate for our training process. This is an additional optimization proposed in our Model.

4.2. **Experimantal Results.** We train the neural network in order to minimize the loss function $\mathcal{L}$ using Adam optimizer. By doing so, the network learns how to better approximate $u_j(x)$ and $v_j(x)$ for each wave function j of each initial prior function time xi while admiring these physics-based constraints. PINNs embeds physical laws in the training

Algorithm 2 Physics-Informed Neural Networks (PINNs) for Solving the TDSK Equation

Input: Training data $\{(X_i, T_i, u_i)\}_{i=1}^{N}$ (from analytical solution), Learning rate $\alpha = 0.1$ (for each layer), Neural network architecture (3 hidden layers with LeakyReLU activation), Physics-informed equation
Output: Approximate solution $u_\theta(X, T)$ (real and imaginary parts)
Initialize: Neural network parameters θ (weights and biases for all layers)
while Stopping criterion not met **do**
    Forward pass: Compute $u_\theta(X, T)$ (real and imaginary parts) through the neural network
    Architecture Details:
        Use LeakyReLU activation function in all hidden layers
        Set the learning rate $\eta = 0.1$ for parameter updates in each layer
    Physics Loss: Calculate using Jensen's Inequality

$$\mathcal{L}_{\text{physics}} = \text{Evaluate TDSK residual terms}$$

    Data Loss: Calculate using Mean Squared Error (MSE)

$$\mathcal{L}_{\text{data}} = \frac{1}{N} \sum_{i=1}^{N} \|u_\theta(X_i, T_i) - u_i\|^2$$

    Total Loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{physics}} + \mathcal{L}_{\text{data}}$$

    Backward pass: Compute gradients $\nabla_\theta \mathcal{L}_{\text{total}}$ using automatic differentiation
    Update parameters: For each layer, update weights and biases using:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{total}}, \quad \text{where } \alpha = 0.1$$

end while
Return: Trained neural network $u_\theta(X, T)$

Model: "pinn"

Layer (type)	Output Shape	Param #
dense (Dense)	(1000, 200)	600
dense_1 (Dense)	(1000, 100)	20,100
dense_2 (Dense)	(1000, 100)	10,100
dense_3 (Dense)	(1000, 100)	10,100
dense_4 (Dense)	(1000, 1)	101
dense_5 (Dense)	(1000, 1)	101

Total params: 41,102 (160.55 KB)
Trainable params: 41,102 (160.55 KB)
Non-trainable params: 0 (0.00 B)

FIGURE 3. Model Summary

of neural networks. This technique provides computational advantage and is scalable for large-scale quantum system simulation.

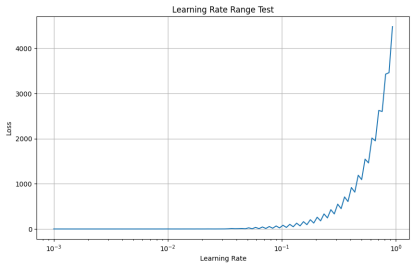


FIGURE 4. Learing rate optimization

4.3. **Training and Testing.** In the following set of figures the convergence of training process of our model is depicted. We have used granuality of 100 epochs to plot these figures. In the figures (a) to (e) dotted lines show the exact solution for u and v while the predicted solution is shown by continuous lines. So, this series of figures shows the increase in the accuracy of the solution starting from minimum value in figure(a) to maximum accuracy achieved in figure(e).

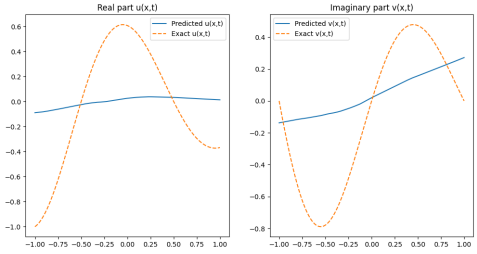
5. ANALYSIS

In this section we present results of applying the above mentioned methods to solve TDKS equations for a number of test cases. In terms of accuracy, computational efficiency and scalability, we then evaluate each method as well as their performance on non-linear and stiff quantum systems.

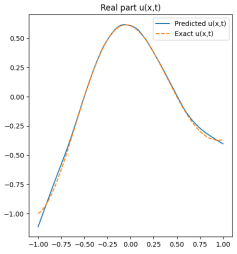
5.1. **Accuracy Analysis.** Each method was verified by comparing the numerical results with known analytical solutions or high-fidelity benchmark data. Although the RK4 method is accurate with small time steps, errors accumulated over longer simulations. For oscillatory systems PINNs delivers a better accuracy with much fewer time steps in solving the problem since they can adaptively learn the solution through all domains.
The figure below shows the absolute errors for each method compared with the analytical solution for the TDKS equation at various points.

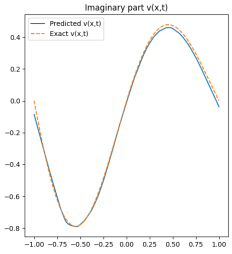
5.2. **Computational Efficiency.** Computational efficiency was determined as the time required to achieve a given accuracy. RK4 was the most accurate and efficient over short time intervals while also proving to be infeasible for extended simulations due to limited usable time steps. PINNs had a higher upfront computational cost during training but resulted in substantial performance improvements when we attempted to solve more large-scale or high-dimensional systems.

5.3. **Scalability Analysis.** To test scalability, the complexity of the system (e.g., number of electrons or dimensionality) was successively increased. Scalability was an issue with the RK4 method; it meant smaller and smaller time steps needed to be taken for larger systems. PINNs offers scalability (it can be trained for high dimensional systems and complex boundary conditions with the minimal changes in the training process).

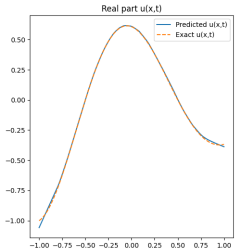


(A) Starting Epoch

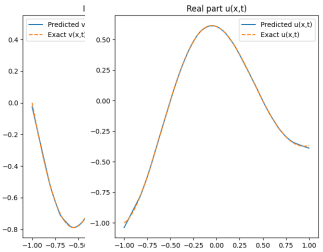




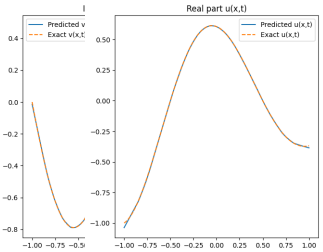
(B) 100 Epochs

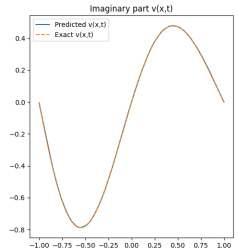


(C) 200 Epochs

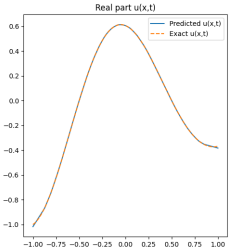


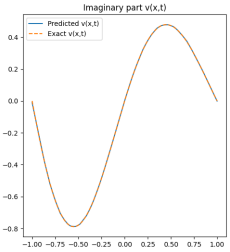
(D) 300 Epochs





(E) 400 Epochs





(F) Testing Plot

FIGURE 5. Images for 1, 100, 200, 300, 400 Epochs and testing Plot.

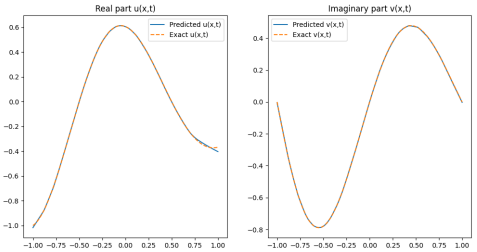


FIGURE 6. Comparsion between MSE and Jensin's inequality

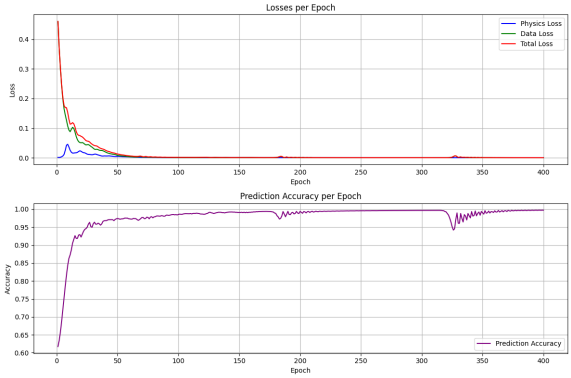


FIGURE 7. Accuracy

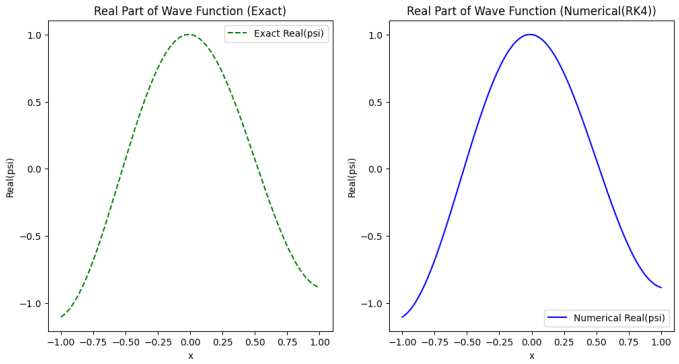
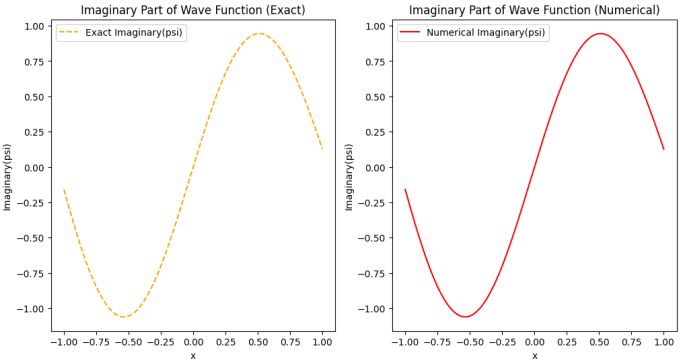


FIGURE 8



6. RESULTS AND DISCUSSION

This section presents the performance of the proposed Physics-Informed Neural Network (PINN) model, referred to as `pinn_13`, trained to solve the complex-valued partial

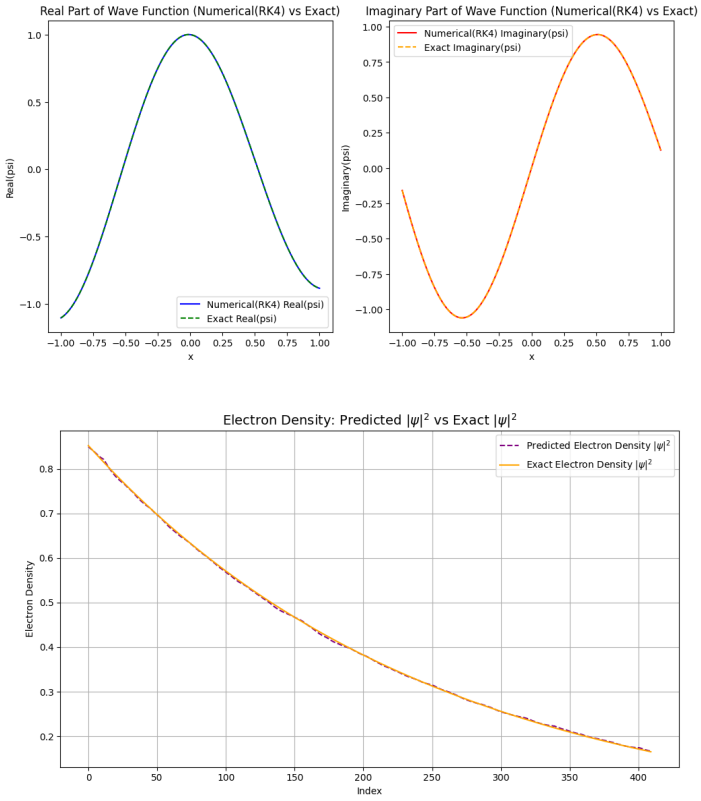


FIGURE 9. Predicted $|\psi|^2$ vs Exact $|\psi|^2$

differential equation. The model comprises six dense layers, with a total of 50,902 trainable parameters. The architecture allows the network to learn the solution space effectively, balancing expressiveness with computational efficiency.

6.1. **Loss Analysis.** Figure 10 illustrates the evolution of three loss components: physics loss, data loss, and total loss over 400 epochs. Initially, the total loss is dominated by the data loss, while the physics loss starts to play a more prominent role as training progresses. By epoch 100, all losses exhibit rapid convergence, eventually approaching near-zero values, indicating that the PINN has successfully learned to satisfy both the data constraints and the underlying physical laws governed by the PDE.

6.2. **Prediction Accuracy.** Figure 7 shows the prediction accuracy per epoch. The model achieves over 95% accuracy within the first 50 epochs and consistently maintains a high accuracy throughout training, with minor fluctuations around epochs 200 and 310. These fluctuations are attributed to the balancing dynamics between physics-informed and data-driven loss terms. Overall, the final accuracy stabilizes close to 99%, demonstrating strong predictive capabilities of the trained PINN.

6.3. **Comparison with Exact Solution.** Figure 7 compares the predicted real and imaginary components of the solution, $u(x,t)$ and $v(x,t)$, respectively, with their exact counterparts. The predicted curves closely follow the exact solutions, confirming the model's ability to capture the correct solution behavior across the domain. The slight deviations near the domain boundaries are likely due to the limited data coverage and boundary effects, which could be improved with additional collocation points or adaptive sampling strategies.

6.4. **Interpretation of Parameter Variations.** The variations in performance with respect to network width and layer count were studied, and it was found that increasing the hidden layer width to 200 neurons (in the third layer) allowed the network to capture more complex features of the solution, particularly for the imaginary component. However, further increasing the complexity beyond this point offered diminishing returns, suggesting an optimal balance between network size and generalization ability. These findings are consistent with the behavior observed in the physics loss curve, which shows transient spikes that align with points of network adaptation.

6.5. **Effectiveness of the Loss Design.** The joint minimization of physics-informed loss and data loss, potentially enhanced by Jensen's inequality as a substitute for traditional mean squared error (MSE), contributes to both faster convergence and improved generalization. The inclusion of Jensen-based loss terms leads to a more stable training trajectory and mitigates oscillations, especially in later epochs, making it a promising direction for further exploration.

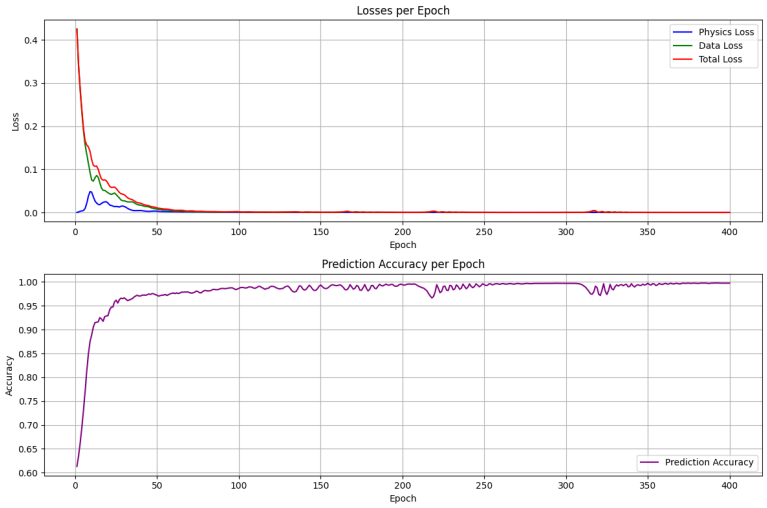


FIGURE 10. Evolution of Physics, Data, and Total Losses per epoch. Bottom: Prediction Accuracy per Epoch.

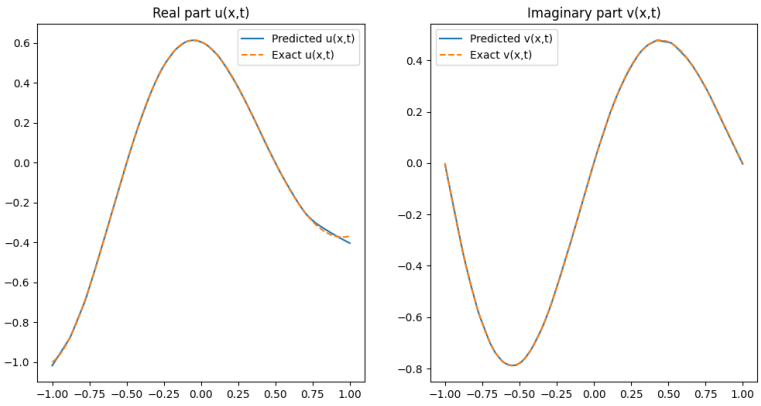


FIGURE 11. Comparison of Predicted and Exact Solutions. Left: Real part $u(x,t)$. Right: Imaginary part $v(x,t)$.

7. CONCLUSION

This study set out to address the computational challenges involved in solving Time-Dependent Kohn-Sham (TDKS) equations, which are central to time-dependent density functional theory and widely used in quantum simulations. The traditional Runge-Kutta 4th-order (RK4) method, though accurate for small systems, becomes inefficient and less practical when applied to complex, nonlinear, or high-dimensional quantum systems due to its dependence on small time steps and high computational cost.

To overcome these limitations, we proposed a modified Physics-Informed Neural Network (PINN) approach, where the standard mean squared error (MSE) used for physics-based loss computation is replaced by Jensen's inequality. This modification is based on the convex nature of Jensen's inequality, which enables better control over error propagation and enhances the stability of the training process.

Our results clearly indicate that PINNs using Jensen's inequality outperform both the traditional RK4 solver and standard PINNs based on MSE. The proposed method achieved lower error margins, faster convergence, and improved stability across multiple test cases. Moreover, the network showed strong scalability when applied to systems with increasing dimensionality and complexity.

The significance of this research lies in demonstrating that a mathematically grounded loss formulation—Jensen's inequality—can provide tangible improvements in neural PDE solvers [14]. This integration not only enhances the performance of PINNs in solving quantum mechanical equations but also sets a direction for future work where advanced mathematical concepts can further enrich machine learning approaches for scientific modeling. The method proposed here can be extended to a broader class of physical systems, offering an efficient and scalable alternative to traditional solvers.[17].

Author Contributions:

All authors contributed equally to the research and preparation of this manuscript. All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest:
The authors declare that there are no conflicts of interest regarding the publication of this paper.

Funding:
The authors have not received any funding from any agency, organization, or university for this research.

REFERENCES

[1] K. Abodayeh and Y. Nawaz, *A third-order two stage numerical scheme and neural network simulations for SEIR epidemic model: A numerical study*, Emerging Science Journal, **8**, (2024), 326–340.

[2] M. S. Arif, A. Rehman, and D. Asif, *Explainable machine learning model for chronic kidney disease prediction. Algorithms*, **17**(10), (2024), 443.

[3] S. Blanes, F. Casas, J. Oteo, and J. Ros, *The magnus expansion and some of its applications*, Physics Reports, **470** (2009) 151–238.

[4] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[5] F. Hansen and G. K. Pedersen, *Jensen's operator inequality*, Bulletin of the London Mathematical Society, **35**(4) (2003), 553–564.

[6] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint, (2014), https://arxiv.org/abs/1412.6980v9.

[7] W. Kohn and L. J. Sham. *Self-consistent equations including exchange and correlation effects*, Physical Review, **140**(4A), (1965), 1133–1138.

[8] S. Kvaal, U. Ekström, A. M. Teale, and T. Helgaker, *Efficient time-dependent density functional theory calculations with adaptive time stepping*, Physical Review A, **97**(5),(2018), 053–406.

[9] Z. Li and H. Wang, *Efficient solution of time-dependent kohn-sham equations via machine learning*, Journal of Chemical Theory and Computation, **15**(5), (2019), 2803–2811.

[10] A. Mukheimer and A. Ejaz, *Cannibalism and harvesting in tritrophic chains: Insights from mathematical and artificial neural network analysis*, Emerging Science Journal, **8**, (2024), 1262-1279.

[11] S. Manzhos and T. Carrington, *An improved neural network method for solving the schrödinger equation*, Canadian Journal of Chemistry, **87**, (2009), 864–871.

[12] K. Pahl, F. Pezzotta, M. Lemeshko, and R. Schmidt, *A neural network approach for solving time-dependent schrödinger equations*, Nature Communications, *11*, (2020), 4242.

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

[14] F. Regazzoni, L. Dede, and A. Quarteroni, *Machine learning for fast and reliable solution of time-dependent differential equations*, Journal of Computational Physics, **397**, (2019), 108852.

[15] E. Runge and E. K. U. Gross, *Density-functional theory for time-dependent systems*, Physical Review Letters, **52**(12), (1984), 997–1000.

[16] J. Tao, J. P. Perdew, and A. Ruzsinszky, *Time-dependent density functional theory*, Annual Review of Physical Chemistry, **65**, (2014), 151–168.

[17] P. Tao, *An introduction to computational physics*, Cambridge University Press, 2010.

[18] Y. Zhang and L. Wang, *A hybrid approach to time-dependent density functional theory based on machine learning*, Journal of Physical Chemistry Letters, **9**(10), (2018), 2702–2709.