ELECTRICAL ENGINEERING AND ENERGY

OPEN ACCESS

Warm-starting conventional solver for ACOPF prediction using Modified Seagull Optimization Algorithm and physics-informed spatiotemporal graph neural network

Abdul-Majid Issah Malori[1], Emmanuel Asuming Frimpong[1], Francis Effah Boafo[1], Elvis Twumasi[1], Peter Asigri[1], and Bernard Marfo Adjei[1]

[1] Department of Electrical and Electronic Engineering, Kwame Nkrumah University of Science and Technology, PMB University Post Office. Kumasi, Ashanti Region, Ghana

Submitted: 6 May 2026
Accepted : 5 June 2026
Online First: 10 June 2026

Corresponding author
Abdul-Majid Issah Malori,
issahmalori@bolgatu.edu.gh

DOI: 10.64470/elene.2026.34

RESEARCH ARTICLE

Copyright, Authors, Distributed under Creative Commons CC-BY 4.0

Abstract: The penetration of renewable energy has intensified the computational challenges associated with solving the AC Optimal Power Flow (ACOPF) problem. The Newton–Raphson (NR) solver is highly sensitive to initialization and may exhibit slow convergence under dynamic conditions. This research proposes a physics-informed spatiotemporal graph neural network (PIStGNN) framework to generate initial conditions for warm-starting the NR solver to improve computational efficiency. The model integrates graph convolutional networks to capture system topology and recurrent architectures (LSTM/GRU) to model temporal variability. A physics-informed loss function enforces power flow constraints, while a Modified Seagull Optimization Algorithm (MoSOA) enhances hyperparameter tuning. The proposed method is compared with DC and flat start warm-start strategies for IEEE 33-, 57-, and 118-bus systems. The results show that the PIStGNN method maintains high accuracy, with MAE values of 0.0126, 0.0446, and 0.0063 for IEEE 33-, 57-, and 118-bus systems respectively. The proposed PIStGNN initialization maintains the lowest runtime across all test systems. On average, the proposed method achieved 1.3–18% computational time reduction compared to conventional initialization techniques. Overall convergence is faster even with more iterations. The high-quality starting point makes the path to convergence more stable. The physics-informed learning provides efficient and scalable ACOPF solutions.

Keywords AC optimal power flow, Modified seagull optimization algorithm, Spatiotemporal graph neural network, Warm starting.

1. Introduction

Modern power systems are experiencing a deep transformation as a result of uncertainty in both demand and supply sides. The integration of renewable energy sources (RESs introduces intermittency and variability (Tudoras-Miravet, Gonzalex-Iakl, Gomis-Bellmunt, & Prieto-Araujo, 2024). The optimal power flow (OPF) problem is at the centre of power system studies. It is used in both short-term and long-term power system planning (Khaloie,et al, 2025). To maintain the security of power systems, security constraints

OPF (Berizzi, et al., 2021) known as Transient Stability Constrained is commonly explored. It envisions transmission line and generator contingencies. It also introduces power flow constraints to the problem. This leads to intractable optimization problems that cannot be solved in computation times attainable for real life application. With this challenge, grid operators use Direct Current OPF (DCOPF), which is less computationally expensive (Guha, et al, 2019). However, the linear assumptions imposed by DCOPF, increase the likelihood of instability and grid failure (Frank & Rebennack, 2016). The use of DCOPF also has impact on climate change. The Federal Energy Regulatory Commission estimated that the approximate techniques cost billions of dollars and cause carbon emissions due to computational inefficiencies (Ahmad, et al., 2022).

In addition, analytical approaches are limited in scale and complexity of ACOPF problem. Beyond the challenges of nonlinearity, the growing diversity of uncertainty sources make conventional methods inadequate (Baker, 2019). The use of warm start to initialize the iterative procedure usually a Newton Raphson method ensures convergence. The major challenge is the choice of initial voltage magnitude and angle guess for the Newton based methods. It has been confirmed in literature that Newton Raphson is very sensitive to initial conditions (Casella & Bachmann, 2021) (Sun, et al, 2017). When the initial guesses are far from the convergence region, the number of iterations increases (Deng & Chiang, 2013). In practice, flat start, and DCOPF solutions, are used to initialize the voltage magnitude and angles (Wang & Tan, 2022) (Vyakaranam, et al, 2021). However, a flat start uses the previous timestep's solution which may not be a good approximation of the current solution. Especially as more intermittent renewable energy sources and inverter-interfaced assets become grid connected (Baker, 2019). This may lead to extra computational time. On the other hand, machine learning (ML) technique offers a good initialize guess to Newton-Raphson power flow. İt can map a dispatch case to the voltage solutions and once sufficiently trained, it requires little computational time to predict its solution.

In the context of ACOPF, machine learning approach utilizes abundant historical data to train an offline process. İt enables rapid direct solution prediction or acceleration of traditional ACOPF optimizers. There are two known ML approaches for OPF evaluation namely; End-to-End (E2E) learning and Learning-to-Optimize (L2O) approaches. The E2E learning approach directly derived OPF solution from inputs without explicitly solving the OPF optimization problem. Despite the progress data-driven methods have made in power system modeling (Bassey, 2023), it suffers the limitations of poor interpretability. It often treat the system as a "black box", without explaining underlying mechanisms, which can lead to hard-to-interpret results (Lin, et al, 2025). It also lack the ability to generalize system states that are not well reflected in training data. Conversely, L2O approaches leverages the advantages of machine learning and traditional optimizer to improve interpretability. In (Okhuegbe, et al, 2024) ML initialize Newton-Raphson power flow by predicting the initial voltage/angle. The initializer successfully converged 2,106 power flow cases which failed to converge due to bad initialization. Reference (Deihim, et al, 2024) warm-started ACOPF solvers using GNN-based approach. It achieved up to 3.75× faster convergence and maintained solution quality. Cao et al. (Cao, et al., 2023) used multi-target binary decision tree with post-pruning to warm-start point learning method for ACOPF. It generated interpretable and feasible initializations that accelerate solver convergence. Almost all existing works are agnostic to the power grid topology. Whenever the grid topology and other operation conditions change in daily operations, there is the need to re-train the model. The lack of topology adaptivity affects its adoption by grid operators. The feasibility issue of OPF ML solutions is very important, especially for the transmission or distribution network constraints.

Lastly, metaheuristic optimization algorithms have shown remarkable promise in solving nonlinear, non-convex, high-dimensional optimization problems. The Seagull Optimization Algorithm (SOA) has gained interest because of its straightforward design, minimal reliance on parameters, and powerful exploratory ability in (Dhiman & Kumar, 2019) (Chen, et al, 2024). The review of SOA approach in (Saini & Rahi, 2024) shows its capability to capture intermittency of sustainable power systems. Using SOA,

reference (Paramguru, et al, 2022) reduced the cost of power generation in a microgrid powered by RES. Although the algorithm showed competitive results, it had trouble managing wind generation uncertainty, which calls for hybrid approaches. A MOSOA-TOPSIS hybrid approach for Combined Cooling, Heating, and Power (CCHP) systems was presented in another study (Li, et al, 2021). Although the hybrid approach increased emissions and energy efficiency, more research was necessary for wider commercial use. In another work, the multi-objective Modified SOA for environmental economic dispatch across IEEE 30-, 57-, and 118-bus systems was proposed in the paper. A Double Group Evolutionary SOA (DSOA) for estimating photovoltaic (PV) cell parameters was proposed in the paper (Gonggui, et al., 2023). Although DSOA increased accuracy and stability, it was limited by its computational complexity. The Rat Swarm Optimizer (RSO) and SOA were compared in the paper (Wei, et al, 2022,) for unit commitment problems in power systems with RES. Although SOA reduced generation costs, it was less effective in situations with a lot of uncertainty due to its deterministic nature. An SOA-RBFNN hybrid was suggested in paper (Oda, et al, 2022) for smart grid energy management. The results demonstrated improved load forecasting and operational efficiency. Despite its merits, SOA's limited exploitation and premature convergence needed adaptive improvements. Addressing these limitations, this study advances the central premise that integrating physics-informed spatiotemporal graph learning with adaptive metaheuristic hyperparameter optimization can substantially enhance ACOPF prediction performance in renewable-dominated power systems. In order to overcome these limitations, this study proposes a Modified Seagull Optimization Algorithm (MoSOA) that combines three adaptive mechanisms.

i. an adaptive weight factor to dynamically balance exploration and exploitation;
ii. a chaotic perturbation-driven strategy incorporating four decay functions (linear, cosine, quadratic, and exponential); and
iii. a hyperbolic tangent–based helical coefficient to speed up convergence and enhance search precision.

All of these changes work together to prevent local minima, increase population diversity, and guarantee strong global search behavior. For ACOPF prediction on IEEE 33-, 57-, and 118-bus systems, MoSOA is incorporated into a physics-informed deep learning pipeline for hyperparameter optimization of PIStGNN, in addition to benchmark evaluation. MoSOA's incorporation into deep learning training loops guarantees adaptive control over learning rates, dropout rates, and network depths, leading to superior constraint satisfaction and faster convergence than more conventional optimizers.

This research aims to address the challenges posed by the integration of renewable energy sources into the power grid, which introduces variability and uncertainty into the power flow problem. The physics-informed spatiotemporal graph neural networks (PIStGNNs) enhance the Newton-Raphson method to improve the feasibility and convergence. In power systems, physics-informed OPF learning framework have been proposed in in (Huang & Wang, 2023) (Qin, Yang, & Yu, 2025), where grid topology is embedded by representing buses as nodes and transmission lines as edges naturally aligning the architecture with the physical system structure. To achieve the aforementioned objective, these central research questions are investigated: How can we significantly improve the computational efficiency and success rate of the optimization algorithm for ACOPF in power systems? What is the impact of incorporating physical constraints into the loss function? This research introduces a new way to achieve ACOPF convergence in large power grids by estimating the initial voltage magnitude and angles using physics-informed spatiotemporal graph neural network. The proposed technique combines dynamic graph convolution networks (GCNs) with advanced recurrent architectures, such as long short-term memory (LSTM) or gated recurrent units (GRU). The dynamic GCN layers are designed to capture the electrical topology using adjacency matrix in the graph convolution operation. The LSTM/GRU layers learn variability from renewable generation, load variations, and operational fluctuations. The physics-informed spatiotemporal graph neural network initializer was applied to standard IEEE 33-bus, 57-bus, and 118-bus systems using Pandapower under various operating conditions. The rest of this paper is structured as follows. Section 2

presents the used model of a generic electrical power system. Explains our physics-informed neural network architecture and training design. It also describes our data collection, simulations, and evaluation methodology for enhancing the network performance and choosing hyperparameters. Section 3 presents a detailed analysis of our computational characterization and validation results. Section 4 discusses the implications of the study

2. Material and Method

2.1 Formulating ACOPF Problem

Given that the power system network is modeled as graph network with n buses and m edges. For each bus i, voltage magnitude (V_i), voltage angle (θ_i), active power generated (P_i^G), reactive power generated (Q_i^G) , active load (P_i^D) and reactive load (Q_i^D). P_{ij}^f and Q_{ij}^f also denote the active and reactive power flowing from bus i to bus j. The admittance between buses i and j is $G_{ij} - jB_{ij}$. θ_{ij} represent the shorthand for $\theta_i - \theta_j$. In this context buses are categorized into slack (S), generator (PV), and load (PQ) nodes to capture their specific operational roles. The ACOPF problem is formulated as shown below:

$$\min_{V,\theta} F(X) \quad (1)$$

s.t.

$$W_{Qi} = V_i \sum_{j=1}^{N} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) - Q_i^G + Q_i^D = 0, \quad (3)$$

$$W_{pMi} = V_i \sum_{j=1}^{N} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) - P_{Gmax} \leq 0, \quad (4)$$

$$W_{pNi} = V_i \sum_{j=1}^{N} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) - P_{Gmin} > 0, \quad (5)$$

$$W_{QMi} = V_i \sum_{j=1}^{N} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) - Q_{Gmax} < 0, \quad (6)$$

$$W_{QNi} = V_i \sum_{j=1}^{N} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) - Q_{Gmin} \geq 0, \quad (7)$$

$$W_{vMi} = V_i - V_{max} \leq 0, \quad (8)$$

$$W_{vNi} = V_i - V_{min} \geq 0, \quad (9)$$

$$W_{PMl} = P_l - P_{lmax} \leq 0, \quad (10)$$

$$W_{PNl} = P_l - P_{lmin} \geq 0 \quad (11)$$

The augmented objective function is constructed combining the constraints (2)– (11) with the objective function (3.1) with penalty factors.

$$L(X) = F(X) + \sum_{i=1}^{N} r_{pi} W_{pi}^2(X) + \sum_{i=1}^{N} r_{Qi} W_{Qi}^2(X) + \sum_{i=1}^{N} r_{vi} W_{vi}^2(X) + \sum_{l=1}^{Nl} r_{pl} W_{pl}^2(X) \quad (12)$$

where
X: the vector that consists of V and θ
$W_{pi}(X)$= active power mismatch constraint
$W_{Qi}(X)$ = reactive power mismatch constraint,
$W_{Vi}(X)$= voltage constraint violation,
$W_{pl}(X)$= transmission line constraint violation,
r_{pi}, r_{Qi}, r_{vi}, r_{pl}= penalty factor
N: the total number of buses

The OPF problem in equation (12) is an unconstrained optimization problem. The constraints violation is captured as the penalty factor. The penalty factor will be zero if the constraint is not violated. The unconstrained optimization problem is solved by the Newton method.

2.1.1 Graph Convolution Network Block Based on Physical Grid Topology

The structures of electric power systems constantly change as a result of maintenance operations, the integration of renewable energy sources, equipment failures, and outages. A dynamic graph, in which the nodes (buses) and edges (lines) vary over time is an efficient way to represent this power system. To reflect the changes in connections and node attributes, the adjacency matrix and feature matrices increase in dimension as the graph moves from $G_{t-1} = (V_{t-1}, E_{t-1})\ to\ G_t = (V_t, E_t)$. As shown in Figure 3 the evolution of the transmission network between two-time steps. While red edges in G_t represent reconfigured line introduced to optimize operational goals, blue edges in G_{t-1} indicate stable connections. These topological shifts are correspondingly mathematically encoded by the adjacency matrices A_{t-1} and A_t . Red cells indicate newly created connections during reconfiguration, while blue cells indicate active lines. The topology may change over time due to switching operations, contingencies, or network reconfiguration. These structural changes are captured through the time-varying adjacency matrix $A_t \in \mathbb{R}^{N\times N}$. where N is the number of buses and Aij=1if buses i and j are electrically connected. The proposed model strictly uses the physical power grid topology derived from the current network configuration. Therefore, message passing occurs only along actual transmission lines. The normalized adjacency matrix used for graph convolution is defined as

$$\tilde{A}_t = D_t^{-\frac{1}{2}}(A_t + I)D_t^{-\frac{1}{2}} \tag{13}$$

where D_t is the degree matrix of A_t and I is the identity matrix introducing self-loops. The graph convolution operation at layer l is then expressed as

$$H_t^{(l+1)} = \sigma\left(\tilde{A}_t H_t^{(l)} \mathrm{W}(l)\right) \tag{14}$$

where

- $H_t^{(l)}$ represents the node embeddings at layer *l*,
- $W(l)$ denotes the trainable weight matrix, and
- $\sigma(\cdot)$is a nonlinear activation function such as ReLU.

The initial layer uses the raw node features: $H_t^{(0)} = X_t$. After L graph convolution layers, the spatial representation becomes $S_t = H_t^{(L)}$. Which encodes topology-aware electrical interactions among buses. This formulation aligns with the implementation based on PyTorch Geometric's GCNConv operator, where graph message passing occurs strictly along the actual transmission network topology determined by the current breaker and switch states.

2.1.2 Long short-term memory block

Long Short-Term Memory (LSTM) networks are well suited for time-series prediction in dynamic systems. Therefore, highly appropriate for sustainable power system operation analysis. Power grids with high renewable penetration exhibit strong temporal characteristics. Capturing both short-term fluctuations and long-term operational trends is essential for accurate spatiotemporal modeling of ACOPF states. At the core of the LSTM architecture lies the memory cell, which functions as an information accumulator. It selectively retains or discards temporal knowledge over extended time horizons. This selective memory control is achieved through three gating mechanisms that regulate information flow across time steps.

1) Input Gate: determines which components of the current input should be written into the memory cell.
2) Forget Gate: decides which information from the previous cell state should be removed.

3) Output Gate: regulates how the updated cell state contributes to the hidden state and network output.

Through these gating operations, LSTMs effectively mitigate the vanishing and exploding gradient problems common in conventional recurrent neural networks, thereby ensuring stable gradient propagation during training. The input vector at time step t is denoted as X_t, while h_{t-1} and c_{t-1} represent the previous hidden and cell states, respectively. These states are updated to the current hidden stateh_t and cell state c_t through nonlinear transformations governed by sigmoid (σ) and hyperbolic tangent $(tanh)$ activation functions. The internal gating structure of the LSTM unit, comprising the forget, input, and output gates that collectively regulate temporal information flow.

1) Forget Gate

The forget gate (Fig. 5a) determines which historical information should be discarded from the previous cell state. It receives the current input X_t and previous hidden state h_{t-1} and produces a gating vector:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (15)$$

where (σ) outputs values in $(0,1)$, representing the degree of information retention. Values close to zero indicate forgetting, while values near one preserve past information.

2) Input Gate

The input gate controls the incorporation of new information into the cell state through a two-stage process:

First, a sigmoid layer determines which elements will be updated:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (16)$$

Second, a candidate memory vector is generated:

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (17)$$

The cell state is then updated by combining retained historical information and new candidate content:

$$c_t = f_t^i \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (18)$$

3) Output Gate

The output gate regulates the information propagated to the hidden state and subsequent time steps:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (19)$$
$$h_t = o_t \odot \tanh(c_t) \quad (20)$$

This mechanism ensures that only relevant components of the updated memory cell influence the network output.

2.1.3 Gated Recurrent Unit (GRU) block

This study also considers GRU, a computationally efficient recurrent architecture designed to capture temporal dependencies using a simplified gating mechanism. The GRU merges memory and hidden representations into a single hidden state unlike LSTM. This reduction in structural complexity often yields faster training and lower parameter count. This can be advantageous for real-time ACOPF surrogate

learning. Given input X_t and previous hidden state h_{t-1}, the standard GRU equations are defined as:

update gate $z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$ (21)

reset gate $r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$ (22)

candidate hidden state $\tilde{h}_t = tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$ (23)

hidden state update $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$ (24)

Where $\sigma(\cdot)$ is the sigmoid function, $tanh(\cdot)$ is the hyperbolic tangent, and $\odot$ denotes elementwise multiplication.

2.1.4 Successive Spatiotemporal GNN (GCN–LSTM)

Let the input sequence over a temporal window of length T be

$$X = \{X_1, X_2, \dots, X_T\}, X_t \in R^{N \times F} \quad (25)$$

where N is the number of buses and F is the node feature dimension. For each time step t, the graph convolution block extracts spatial features using the grid adjacency: $H_t^{(o)} = X_t$

$$H_t^{\ell+1} = \sigma\left(\tilde{A}_t H_t^{(\ell)} W^{\ell} + b(\ell)\right), \qquad \ell = 0,1, \dots, L-1 \quad (26)$$

where $\tilde{A}$ is the normalized adjacency matrix, $W(\ell)$ and $b(\ell)$ are trainable parameters, L is the number of graph convolution layers, and $\sigma(\cdot)$ is the nonlinear activation function. The final spatial embedding at time step t is $S_t = H_t^{(L)} \in \mathbb{R}^{N \times D}$ where D is the spatial embedding dimension. Since the recurrent block in the implementation uses a standard PyTorch LSTM, the spatial embedding is reshaped into a vector sequence before temporal modeling:

$$z_t = vec(S_t) \in \mathbb{R}^{N \times D} \quad (27)$$

The temporal dynamics are then modeled sequentially using a standard LSTM:

$$(h_t, c_t) = LSTM(z_t, h_{t-1}, c_{t-1}) \quad (28)$$

where h_t and c_t denote the hidden state and cell state at time step t, respectively. The final temporal representation is mapped to the output prediction through a fully connected layer:

$$\hat{Y} = W_o h_T + b_o \quad (29)$$

where $\hat{Y}$ contains the predicted ACOPF-related quantities.

2.1.5 Successive Spatiotemporal GNN (GCN–GRU)

For the GRU-based variant, the same spatial encoder is used, and the recurrent stage becomes

$$h_t = GRU(z_t, h_{t-1}) \quad (30)$$

with output prediction $\hat{Y}$, this formulation reflects the actual implementation in which graph convolution is applied. The temporal sequence learning is performed afterward using a standard recurrent unit.

2.1.6 Residual Successive Spatiotemporal GNN

For the residual variants, a skip connection is added between the projected spatial input and the recurrent output. Let the projected residual input be

$$r_t = W_r z_t + b_r \tag{31}$$

Then the residual recurrent representation is $\tilde{h}_t = h_t + r_t$ and the final output becomes $\hat{Y}$. This residual connection preserves input information and improves gradient flow during training.

2.2 Customized Loss Function

The model employs a physics-informed loss function, which is a linear combination of Mean Squared Error (MSE) and a regularization term based on the equality and inequality constraints. The neural network is trained to predict the ACOPF system states while maintaining physical feasibility. The training objective combines supervised prediction accuracy with physics-based regularization terms:

$$L_{total} = w_1 L_{data} + w_2 L_{phys} + w_3 L_{safe} + w_4 L_{branch} \tag{32}$$

where $L_i \in \{L_{data}, L_{phys}, L_{safe}, L_{const}\}$ denotes the individual loss terms. This formulation enables the network to automatically learn optimal trade-offs among tasks with different magnitudes and physical units, thereby improving gradient stability and convergence behavior in ACOPF learning.

1) Data fidelity loss

The data loss enforces supervised consistency between the predicted system state$\hat{y}$ and the ground-truth ACOPF solution y:

$$L_{data} = \|\hat{y} - y\|_2^2 \tag{33}$$

where all variables are normalized to ensure numerical stability. The target vector includes voltage magnitudes, phase angles, and generation dispatch variables. This term ensures that the PIStGNN accurately approximates reference ACOPF solutions generated using conventional solvers.

2) Physics consistency loss

To embed AC power flow physics directly into the learning process, a physics-based regularization term is introduced based on Kirchhoff's Current Law (KCL). The complex power injection at each bus is defined as:

$$S = P + jQ = V \cdot (Y_{bus} V)^* \tag{34}$$

where V is the complex bus voltage vector and Y_{bus} is the bus admittance matrix. The physics loss penalizes deviations between predicted and physically computed power injections:

$$L_{phys} = |P_{net} - Re(V \cdot (Y_{bus} V)^*)|^2 + |Q_{net} - Im(Y_{bus} V)^2|^2 \tag{35}$$

This term constrains the neural network to satisfy nonlinear AC power flow equations, ensuring physical feasibility and reducing the risk of non-compliant predictions commonly observed in purely data-driven models.

3) Operational safety loss

To maintain voltage security within permissible bounds, a soft penalty mechanism is introduced:

$$L_{safe} = ReLU(|V| - V_{max})^2 + ReLU(V_{min} - |V|)^2 \quad (36)$$

where $ReLU(x) = \max(0, x)$ This formulation imposes no penalty when voltage magnitudes remain within limits, while quadratically penalizing violations. The use of soft penalties preserves differentiability and improves training stability compared to hard constraint enforcement.

4) Branch capacity loss (Inequality constraint)

$$L_{branch} = \frac{1}{LB}\sum[ReLU(|S_k| - S_{k,max})^2] \quad (37)$$

where the branch apparent power flow is:$I_R = Y_{branch,k}(V_{from,k} - V_{to,k}), \quad |S_k| = |V_{from,k} \cdot I_k^*|$. $Y_{branch,k}$ is the series admittance of branch $\boldsymbol{k}$, extracted as $Y_{\{\mathbf{bus}\}[\mathbf{from},\mathbf{to}]}$. $S_{k,max}$ is the thermal rating of the branch in per-unit.

2.3 Newton–Raphson Solver

Warm starting the NR solver with PIStGNN has the potential to remove all infeasibilities. The Newton-Raphson method is a standard tool for solving ACOPF problems. The warm-starts for the NR algorithm benefit from initial points. Hence, the PIStGNN model was generalized to predict optimal values. The NR method is a well-established iterative approach to solve these nonlinear algebraic equations efficiently due to its rapid convergence characteristics. The core equations covering the power flow problem are the power balance equations for each bus. For a bus i, the real power P_{ij}^f and reactive power Q_{ij}^f are defined as follows:

$$P_{ij}^f = V_i^2 g_{ij} - V_i V_j(g_{ij}\cos(\theta_{ij}) - b_{ij}\sin(\theta_{ij})) \quad (38)$$

$$Q_{ij}^f = V_i^2 b_{ij} - V_i V_j(b_{ij}\cos(\theta_{ij}) + g_{ij}\sin(\theta_{ij})) \quad (39)$$

where V_i represents the voltage magnitude at bus i, g_{ij} and b_{ij} are the conductance and susceptance of the line between buses i and j, and θ_{ij} is the phase angle difference between buses i and j. The iterative NR method updates guesses for the voltage magnitude and angle at each bus by solving a system of equations derived from the Taylor series expansion of the power balance equations. The Jacobian matrix J, which contains the partial derivatives of the power mismatches with respect to the voltage magnitude and angle, plays a crucial role in this update:

$$J = \begin{bmatrix} \frac{\partial P}{\partial \theta} & \frac{\delta P}{\delta V} \\ \frac{\delta Q}{\delta \theta} & \frac{\delta Q}{\delta V} \end{bmatrix} \quad (40)$$

At each iteration, the power mismatch between the current state and the power specifications is calculated:

$$\Delta P = P_{spec} - P_{calc}(V, \theta) \quad (41)$$

$$\Delta Q = Q_{spec} - Q_{calc}(V, \theta) \quad (42)$$

where, P_{spec} and Q_{spec} are the specified real and reactive power, while P_{calc} and Q_{calc} are the calculated values from the current estimates of voltage magnitude V and phase angle θ. The Jacobian is used to determine the changes ΔV and $\Delta\theta$ to be applied to the voltage magnitudes and phase angles to reduce the power mismatch:

$$\begin{bmatrix} \Delta\theta \\ \Delta V \end{bmatrix} = -J^{-1}\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \quad (43)$$

The voltage magnitude V and phase angle θ at each bus are updated accordingly:

$$\theta_{new} = \theta + \Delta\theta \quad (44)$$

$$V_{new} = V + \Delta V \quad (45)$$

The iterative process continues until the power mismatches ΔP and ΔQ are within acceptable tolerances, indicating convergence to a solution.

2.4 Hyperparameter Optimization Using Modified Seagull Optimization Algorithm

In this work, the MoSOA is enforced to optimally regulate the values of the hyperparameter of the PISGNN method that in turn improves the accomplishment of the regression. In this work, mathematical modelling for global exploration was introduced for longer-distance relocation, and scientific modelling for local development was introduced for spiral outbreaks. The MoSOA mimics the displacement trajectory of a seagull flock when they are flying in the air. MoSOA was selected over other classic optimization algorithms such as Adam and SGD because it achieves a good balance of exploration and exploitation, preventing local optima and allowing for better tuning of the hyperparameters in ACOPF regression tasks. All optimization algorithms aim to minimize or maximize an objective function. In deep learning, this function typically represents a physics informed loss function that needs to be minimized during training.

2.4.1 Mathematical model for Seagull Optimization Algorithm (SOA)

The mathematical models of migration and attack behavior are as follows:

A. Migration (exploration)

The location changes of the seagull populations depend on collisions avoidance between seagulls, the best individual seagull, and location update through the best individual seagull.

1. Collisions avoidance: the position of the individual seagull after it has moved is evaluated using the variable *A*.

$$C_s = A \times P_s(t) \quad (46)$$

where, C_s is the position of the seagull that do not collide with other individuals, P_s represents the current position of the seagull, t is the number of iterations, and A represents the movement behaviors of the seagull, calculated as follows.

$$A = f_c - (t \cdot f_c / Max_{iteration}) \quad (47)$$

where f_c is the control frequency of the manipulation variable A, usually set to 2. $Max_{iteration}$ is the maximum number of iterations. As the variable A decreases from 2 *to* 0 the number of iterations increases.

2. Movement towards best neighbor's direction: the seagull moves towards the optimal individual as expressed below.

$$M_s = B \times (P_{bs} - P_s) \quad (48)$$

where M_s is the new position of the seagull after it has moved towards the optimal seagull bs. $(P_{bs}) \cdot B$ is a random number that is used to balance the exploration and exploitation of the algorithm. It is calculated as follows.

$$B = 2 \cdot A^2 \cdot rand \quad (49)$$

where *rand* is between 0 and 1.

3. Remain close to the best search agent: the equation for the distance the search individual position moves relative to the optimal individual is as follows.

$$D_s=|C_s+M_s| \tag{50}$$

Where D_s is the distance between the searched individual and the optimal individual.

B. Attacking (exploitation)

The attack behavior uses historical experience during the search process and mastered by wings and weight in the direction of the prey. The seagull attacking prey process is generally uses a spiral movement action, which is calculated as follows.

$$\vec{P}_s(t+1)=\left(\vec{D}_s\times x\times y\times z\right)+\vec{P}_{best}(t) \tag{51}$$

$$x=r\cos(k) \tag{52}$$

$$y=r\times\sin(k) \tag{53}$$

$$z=r\times k \tag{54}$$

$$r=u\times e^{kv} \tag{55}$$

where $\vec{P}_s(t+1)$ is the seagull after the position update, r is the spiral radius, k denotes a number between 0 and 2π. The constants that define the spiral action are represented as u and v and e is the base of the natural logarithm. The best seagull individual's algorithm uses a random method to generate the population and updates the gull population position.

C. Multi-strategy Improved Seagull Optimization Algorithm

Adaptive Nonlinear Adjustment of Manipulated Variable A

The seagull population moves toward the best solution while maintaining sufficient distance from neighboring seagulls to avoid collision during the search process. The prey-attacking behavior is modeled by adjusting both the movement angle and flying speed of the seagulls. In the original SOA, the convergence factor (A) controls the balance between global exploration and local exploitation. As defined in Equation (47), the value of (A) decreases linearly from 2 to 0 as the iteration count increases. This gradual reduction strengthens the local search of the algorithm over time. However, because the search behavior of the algorithm is inherently nonlinear. The use of a linearly decreasing weight limits the algorithm's adaptability and may reduce optimization performance. To address this limitation, this study proposes a Modified Seagull Optimization Algorithm (MoSOA), in which the convergence factor is reconstructed to provide a more adaptive balance between exploration and exploitation throughout the optimization process. The mathematical formulation of the proposed modification is presented as follows:

$$A=f_{ada}\cdot\left(1-\sin\left(\frac{\pi}{2}\cdot\frac{t}{Max_{iteration}}\right)\right) \tag{56}$$

is the stochastic adaptive nonlinear strategy introduced by MoSOA, in which the adaptive factor f_{ada} dynamically adapts based on the fitness distribution of the population.

$$f_{c,ada}=f_{c,min}+M\cdot\left(f_{c,max}-f_{c,min}\right)+\sigma\cdot randn \tag{57}$$

As shown in equation (56), A decreases nonlinearly with increase of iterations. It coordinates the global

search and local optimization of the population to improve the algorithm's convergence speed and optimization accuracy.

Adaptive Helical Coefficient

A hyperbolic tangent-based dynamic formulation is used in place of the conventional fixed helical parameters in equation (55) to improve the spiral (attack) phase. This allows for continuous radius contraction as iterations go forward:

$$v=v_{max}\cdot\left(1-\frac{2}{1+e^{\left(\frac{2t}{T_{maxitera}}\right)}}\right)+v_{min} \tag{58}$$

This adaptive coefficient speeds up convergence and prevents stagnation close to local optima by guaranteeing larger spirals for global exploration at first and smaller, tighter spirals for fine local exploitation later.

Improvement of Update Function

The optimization stagnates if the optimal global individual falls into the local optima. To solve this problem, this study uses the learning factor based on equation (51). The process of seagull learning improves through dynamic inertia weight. The attack position update formula with learning strategy is:

D. Dynamic Perturbation Factor and Improved Adaptive Weighting

A nonlinear dynamic weight factor $\omega(t)$ is used to ensure optimal solution is adaptive to the best agent over time.

$$\omega(t)=(\omega_{max}-\omega_{min})\times\left(1-\cos\left(\frac{\pi}{2}\times\frac{t}{T_{Maxitera}}\right)\right)+\omega_{min} \tag{59}$$

MoSOA adds a dynamic perturbation factor $\beta(t)$ into the position update rule in order to maintain diversity: The equations (57)-(59) express linear, cosine, and exponential decay functions respectively are used to adaptively transition the perturbation factor $\beta(t)$. In order to effectively search the solution space and converge toward the global optimum while preserving a healthy balance between diversity and intensification, SOA continuously alternates between migration (exploration) and attack (exploitation).

$$\beta_1(t)=\beta_{max}-\left(\frac{t}{T_{max}}\right)(\beta_{max}-\beta_{min}) \tag{60}$$

$$\beta_1(t)=\beta_{min}+(\beta_{max}-\beta_{min})\times\left(\frac{1+\cos\pi\times\frac{t}{T_{max}}}{2}\right) \tag{61}$$

$$\beta_2(t)=\beta_{max}\times e^{-\lambda\times(t/T_{max})} \tag{62}$$

When taken as a whole, these enhancements allow MoSOA to: (i) Preserve an ideal ratio between local exploitation and global exploration (ii) attain more rapid convergence with greater accuracy; (iii) steer clear of local minima through chaotic perturbation diversity; and (iv) provide consistent performance in nonlinear, multimodal optimization environments.

$$\vec{P}_s(t+1)=\left(\vec{D}_s\times x\times y\times z\right)+\omega(t)*\vec{P}_{best}(t)+\beta(t)\times(\vec{P}_{random}-\vec{P}_s(t)) \tag{63}$$

Over time, the iterations t, gradually reduce the weight to encourage diverse exploration, early iterations favor global guidance ($\omega>1$). where the learning factors $c1$, $c2$ are set to 1.5 and $r1$, $r2$ are random numbers

between [0,1]. ω is the inertia weight, ωmax=0.95, ωmin=0.35, $P\rightarrow GS(t)$ is the direction of the best position in individual history, and $P\rightarrow l(t)$ is the attack position of seagull after learning.

2.4.2 Benchmark set and compared algorithms

The proposed Modified Seagull Optimization Algorithm (MoSOA) is evaluated by comparing its performance with other metaheuristic optimization algorithms. These include the Genetic Algorithm (GA), Gravitational Search Algorithm (GSA), Grey Wolf Optimization (GWO), Particle Swarm Optimization (PSO), Tunicate Swarm Algorithm (TSA), the original Seagull Optimization Algorithm (SOA), and the Improved Seagull Optimization Algorithm (ISOA). The comparison is carried out using 23 standard benchmark functions. Which are grouped into three categories: unimodal, multimodal, and fixed-dimension multimodal functions, as presented in Tables 1-3, respectively. To ensure a fair and reliable evaluation, each algorithm was executed 30 times, and the performance was assessed using four statistical indicators: Best, Worst, Mean, and Standard Deviation (STD). The maximum number of iterations for all experiments was fixed at $(T = 500)$. The benchmark functions were selected to evaluate different optimization characteristics, including convergence speed, solution accuracy, exploration capability, and the ability to avoid local optima. In the benchmark tables, "Dim" represents the dimensionality of the test function, "Range" defines the search interval of the decision variable (x), and (f_{min}) denotes the theoretical optimal value of each function.

Table 1 Details of unimodal benchmark functions

Function	Equation	Dim	Range	Fmin
F1	$f(x) = \sum_{i=1}^{d} X_i^2$	30	$[-100,100]^d$	0
F2	$f(x) = \sum_{i=1}^{d} \lvert X_i \rvert + \prod_{i=1}^{d} \lvert X_i \rvert$	30	$[-10,10]^d$	0
F3	$f(x) = \sum_{i=1}^{d} \left(\sum_{j=1}^{i} X_j \right)^2$	30	$[-100,100]^d$	0
F4	$f(x) = \max_i \{\lvert X_i \rvert,\ 1 \le i \le d\}$	30	$[-100,100]^d$	0
F5	$f(x) = \sum_{i=1}^{d-1} \left[100 \left(X_{i+1} - X_i^2 \right)^2 + (X_i - 1)^2 \right]$	30	$[-30,30]^d$	0
F6	$f(x) = \sum_{i=1}^{d} (\lvert X_i + 0.5 \rvert)^2$	30	$[-100,100]^d$	0
F7	$f(x) = \sum_{i=1}^{d} iX_i^2 + random[0,1))$	30	$[-128,128]^d$	0

2.5 The Proposed Warm-Started Newton–Raphson Solver using Physics informed spatiotemporal graph neural network

Figure 1 illustrates the workflow for warm-starting the Newton–Raphson (NR) solver for ACOPF using PIStGNN techniques. The framework is divided into two main stages. In the first stage, the PIStGNN model is trained using a labeled dataset through supervised end-to-end learning. Once trained, the PIStGNN generates predictions for the voltage magnitudes and voltage angles, which are then used as informed initial guesses for the optimization solver. The generated solutions may not always fully satisfy the physical and operational constraints of the power system. To address these feasibility issues, the study incorporates two post-processing strategies. The first approach involves solving the power flow problem directly within the supervised end-to-end learning framework to improve physical consistency. The second approach uses the PIStGNN predictions to warm-start the exact Newton–Raphson solver. It allows the iterative optimization process to refine the predicted states and obtain a fully feasible ACOPF solution.

Table 2 Details of multi-modal benchmark functions

Function	Equation	Dim	Range	Fmin
F8	$f(x) = -\sum_{i=1}^{d}(X_i \sin(\sqrt{\lvert X_i \rvert}))$	30	$[-500,500]^d$	$-418.982^9 \times n$
F9	$f(x) = 10d + \sum_{i=1}^{d}[X_i^d - 10\cos(2\pi X_i)]$	30	$[-5.12,5.12]^d$	0
F10	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}X_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos 2\pi X_i\right) + 20 + e$	30	$[-32,32]^d$	0
F11	$f(x) = \frac{1}{4000}\sum_{i=1}^{d}X_i^2 - \prod_{i=1}^{d}\cos\left(\frac{X_i}{\sqrt{i}}\right) + 1$	30	$[-600,600]^d$	0
F12	$f(x) = \frac{\pi}{d}\{10\sin(\pi y_i) + \sum_{i=1}^{d-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_d-1)^2\} + \sum_{i=1}^{d}U(X_i,10,100,4)y_i$ $= 1 + \frac{X_i+1}{4}U(X_i,a,k,m)$ $= \begin{cases} k(X_i-a)^m X_i > a \\ 0 - a < X_i < a \\ k(X_i-a)^m X_i < -a \end{cases}$	30	$[-50,50]^d$	0
F13	$f(x) = 0.1\{\sin^2(3\pi X_i) + \sum_{i=1}^{d}(X_i-1)^2[1+\sin^2(3\pi X_i+1)] + (X_d-1)^2[1+\sin^2(2\pi X_d)]\} + \sum_{i=1}^{d}U(X_i,5,100,4)$	30	$[-50,50]^d$	0

Table 3 Details of fixed-dimension multi-modal benchmark functions

Function	Equation	Dim	Range	Fmin
F14	$f(x) = \left[\frac{1}{500} + \sum_{i=1}^{25}\frac{1}{i+\sum_{j=1}^{2}(X_j - X_{j,i})^6}\right]^{-1}$	2	$[-65, 65]^d$	1
F15	$f(x) = \sum_{i=1}^{d}\left[a_i - \frac{X_i(b_i^2+b_iX_2)}{b_i^2+b_iX_3+X_4}\right]^2$	4	$[-5, 5]^d$	0.00030
F16	$f(x) = 4X_1^2 - 2.1X_1^4 + \frac{1}{3}X_1^6 + X_1X_2 - 4X_2^2 + 4X_2^4$	2	$[-5, 5]^d$	-1.0316
F17	$f(x) = \left(X_2 - \frac{5.1}{4\pi^2}X_1^2 + \frac{5}{\pi}X_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos X_1 + 10$	2	$[-5, 5]^d$	0.398
F18	$f(x) = [1+(X_1+X_2+1)^2(19-14X_1+3X_1^2-14X_2+bX_1X_2+3X_2^2)] \times [30+$	2	$[-2, 2]^d$	3
F19	$f(x) = \sum_{i=1}^{4}a_i \cdot \exp(-\sum_{j=1}^{3}b_{ij}(X_j - P_{ij})^2)$	3	$[1, 3]^d$	-3.86
F20	$f(x) = \sum_{i=1}^{4}a_i \cdot \exp(-\sum_{j=1}^{6}b_{ij}(X_j - P_{ij})^2)$	6	$[0, 1]^d$	-3.32
F21	$f(x) = \sum_{t=1}^{5}[(X-a_i)(X-a_i)^T + c_1]^{-1}$	4	$[0, 10]^d$	-10.1532
F22	$f(x) = \sum_{t=1}^{7}[(X-a_i)(X-a_i)^T + c_1]^{-1}$	4	$[0, 10]^d$	-10.4028
F23	$f(x) = \sum_{t=1}^{10}[(X-a_i)(X-a_i)^T + c_1]^{-1}$	4	$[0, 10]^d$	-10.5363

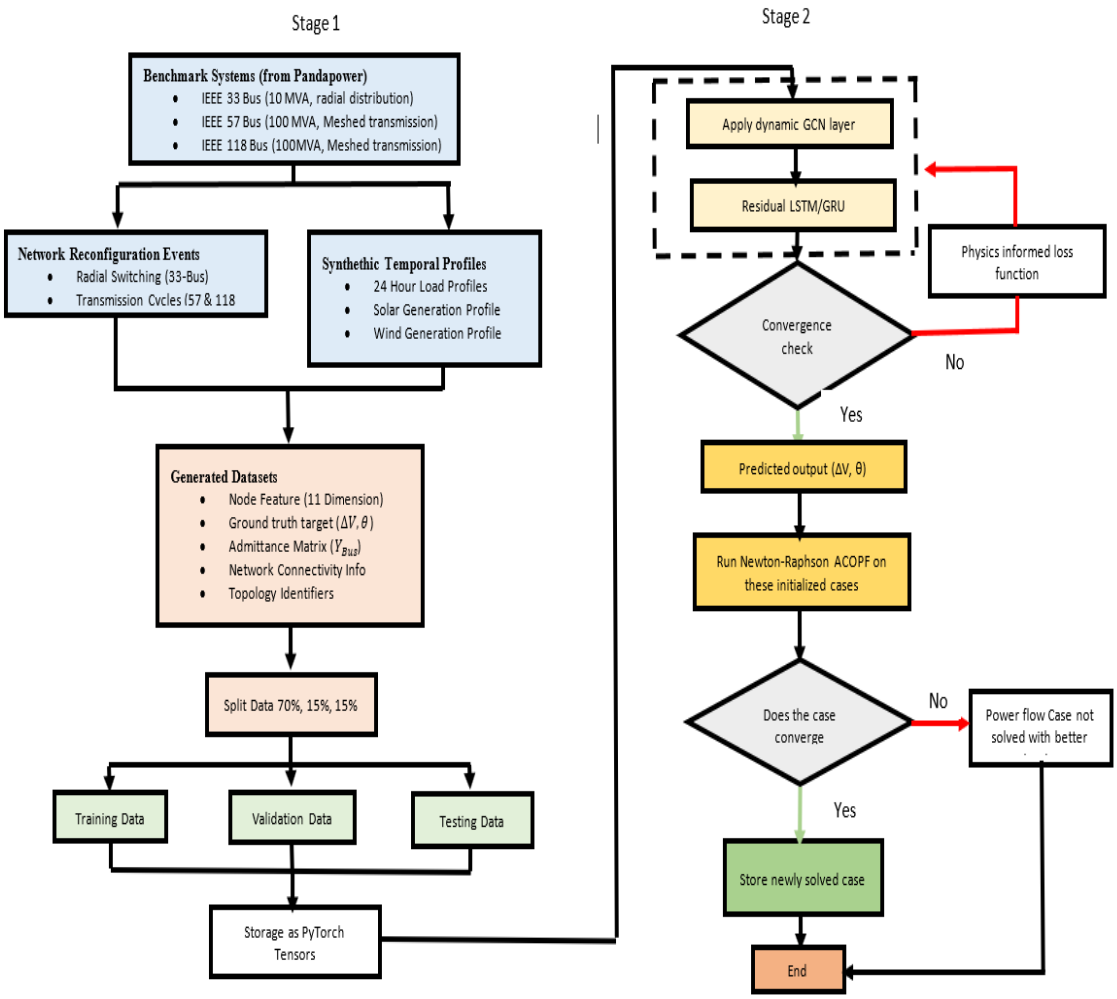


Figure 1 Overall framework of the proposed AC-OPF prediction approach

2.5.1 Data Generation and Preprocessing

The data pipeline generates spatiotemporal power system datasets for training and evaluating the proposed spatiotemporal graph neural network models. Three benchmark systems from the Pandapower library are used: the IEEE 33-bus, IEEE 57-bus, and IEEE 118-bus networks. The use of synthetic temporal profiles enabled controlled evaluation of the proposed method under varying operating conditions. It also avoids issues related to data confidentiality, and inconsistent utility-scale datasets. Nevertheless, the proposed spatiotemporal graph learning architecture remains fundamentally applicable to real operational environments because the model directly incorporates both network topology and temporal system dynamics. The study systems represent distribution and transmission grids with structural complexity. The IEEE 33-bus system is a radial distribution network with normally open tie-lines enabling reconfiguration. The IEEE 57-bus and IEEE 118-bus systems are meshed transmission networks without predefined tie-lines. The base power values are 10 MVA for the 33-bus system and 100 MVA for the 57-bus and 118-bus systems. The 11-dimensional feature vector captures active and reactive load demand, power injections from the external grid, conventional generation, renewable generation, voltage magnitude, voltage angle, and the normalized bus degree representing local connectivity. All power quantities are converted to per-unit values using the system base power. To emulate measurement uncertainty, Gaussian sensor noise is added to the primary electrical variables during data generation. The ground-truth targets are obtained from AC power flow solutions computed in Pandapower. The graph neural networks are trained to predict only voltage

magnitude deviation and voltage angle for each bus. In addition, the temporal operating conditions are generated using synthetic load and renewable generation profiles. The load demand follows a 24-hour "camel-shaped" curve with morning and evening peaks. It has an additional ±5% random perturbation to represent short-term variability. The solar generation follows a bell-shaped daytime curve modulated by seasonal variation and stochastic cloud conditions. While wind generation follows a night-peaking coastal profile governed by weather-dependent wind speed ranges. The weather evolution is modeled using a first-order Markov chain. It enables temporal renewable generation patterns. The data generator also simulates network reconfiguration events. In the radial IEEE 33-bus system, switching occurs by closing a tie-line and opening another line within the resulting loop to maintain radiality. For the meshed IEEE 57-bus and 118-bus systems, cycles are identified and a line is opened to create a temporary tie-line before reconfiguration. The switching configuration is assigned a unique topology identifier, and the corresponding bus admittance matrix (Y_{bus}) is recomputed. The generated data undergo preprocessing before training and are aggregated, converted to per-unit values. The dataset split chronologically into training (70%), validation (15%), and testing (15%) subsets. The generated graph dataset consists of 1000 samples per system and period, which is split into a training set (700 samples), a validation set (150 samples), and a test set (150 samples). The processed datasets, including node features, targets, topology identifiers, admittance matrices, and network connectivity information, are stored as PyTorch tensors to support efficient training and evaluation of the proposed models.

2.5.2 Training in graph neural network architectures

The graph neural network architectures are implemented to evaluate spatiotemporal learning for ACOPF prediction. The models are grouped into spatial-only and spatiotemporal categories and are implemented using PyTorch and PyTorch Geometric. Graph message passing is performed using the GCNConv operator. The node interactions occur strictly along the physical transmission network defined by the grid topology. The spatial-only models process a single system snapshot with input tensor shape (B, N, F). Where B is the batch size, N the number of buses, and F the number of node features. The standardGCN model uses a fixed adjacency matrix corresponding to the base network topology. The dynamicGCN extends this model by updating the adjacency matrix for each sample to reflect the current switching configuration or contingency state. The PIGCN architecture adopts the same structure as dynamicGCN but incorporates physics-informed regularization in the training loss. The spatiotemporal models operate on sequential inputs with tensor shape (B, T, N, F), where T denotes the temporal window length. These models follow a successive spatiotemporal pipeline. For each timestep, graph convolution extracts spatial embeddings $h_{b,t} = GCNConv(X_{b,t}, A_{b,t})$. These are stacked across time and reshaped into per-node sequences. The sequences are then processed using a recurrent neural network to capture temporal dependencies caused by renewable variability and load dynamics. Two variants are considered: PIGCLSTM, which employs an LSTM layer, and PIGCGRU, which replaces the LSTM with a GRU for reduced parameter complexity. To improve representational capacity and mitigate oversmoothing in deeper graph networks, two residual variants are introduced: PIResnetGCLSTM and PIResnetGCGRU. These models employ residual graph convolution blocks consisting of two GCNConv layers with skip connections, allowing stable training of deeper spatial representations. For all spatiotemporal models, the recurrent layer produces a hidden representation from the final timestep of the input sequence. This representation captures the temporal behavior of the power system and is passed through a fully connected layer to predict the network operating state. The final model output has the shape ((B, N, 2)), where (B) represents the batch size, (N) denotes the number of buses, and the two output variables correspond to the voltage magnitude deviation and voltage angle at each bus. To ensure stable and efficient training, several hyperparameter settings were adopted. The models were trained using a batch size of 32, while the Adam optimizer was selected because of its strong convergence properties in deep learning applications. An initial learning rate of 0.001 was used, together with an exponential learning

rate decay strategy to gradually reduce the step size during training. The decay rate was fixed at 0.95, with decay steps set to 5000 iterations. Training was performed for a maximum of 500 epochs, and an early stopping mechanism based on validation loss was implemented to reduce overfitting and improve generalization performance. A patience value of 50 epochs was selected, allowing the model sufficient time to continue learning before training termination. The Mean Squared Error (MSE) loss function was used during training to achieve smooth and stable convergence. MSE measures the average squared difference between predicted and actual target values, enabling effective gradient-based parameter updates through backpropagation and improving overall prediction accuracy.

2.5.3 Detailed Implementation of Modified Seagull Optimization Algorithm

MoSOA is implemented in Python 3.x. and utilizes key libraries such as NumPy and SciPy. MoSOA simulates seagull moving in a spiraling fashion towards the optimal solution while also performing exploration. It updates the positions of a population of 'seagull' iteratively based on the best solution found so far and random exploration components. A population of seagull is initialized randomly within the defined lower bound and upper bound of the search space. The seagull's position represents a candidate solution to the optimization problem. The algorithm proceeds over a predefined number of maximum iterations. In each iteration, the fitness of every seagull is evaluated using the objective function. The best position and its corresponding score found across the entire population up to that point are continually updated. Boundary checks are performed after each position update to ensure that seagull remain within the feasible search space. Any seagull that exceeds the upper bound or fall below the lower bound are clamped back to the respective limit. A linearly decreasing coefficient, $F_{c,ada}$, is employed to balance exploration and exploitation. F_c starts at 2 and linearly decreases to 0 over the maximum iterations. A higher $\mathrm{f_{c,max}}$ in early iterations promotes global exploration, while a lower $\mathrm{f_{c,min}}$, in later iterations encourages local exploitation around promising solutions as shown in equation (34). Adaptive Adjustment (A1) parameter influences the step size towards the best position as shown in equation (33). A time factor (sine function based) and an uncertainty factor, making it more dynamic and responsive to population diversity. A spiral constant, v, controls the tightness of the spiral movement. It varies non-linearly between v_{max}(1) and v_{min} (-1) over iterations expressed by equation (35). This variation allows for a broader spiral search initially and a tighter one as the algorithm converges. A random number (randn) is generated within a range determined by $F_{c,ada}$ to introduce stochasticity into the spiral path. The equations dictate that each seagull i in dimension j updates its position (X1) by moving towards the best position found so far, with a spiraling motion influenced by v and randn. The D_s term represents a scaled distance from the current position to the best position. A major contribution of this work is the introduction of a dynamic perturbation term using beta and a randomly chosen agent's position (P_{rand}). The beta parameter decreases linearly from β_{max} to β_{min} over iterations. This perturbation, $\beta(t) \times \left(\vec{P}_{random} - \vec{P}_s(t)\right)$, as expressed by (40) which adds an additional exploration component by pushing seagull away from their current position towards another random agent, helping to prevent premature convergence. In the Seagull Optimization Algorithm (SOA), the best search agent is expected to exert a stronger influence during the early stages of the optimization process than in the later stages. At the beginning of the search, a larger perturbation coefficient is assigned to the current best individual to accelerate convergence toward the region of the global optimum. This encourages broader exploration of the search space and helps guide the population toward promising solutions more quickly. As the optimization progresses, the perturbation coefficient is gradually reduced to avoid excessively rapid convergence, which may cause the algorithm to become trapped in local optima. By using smaller perturbation values in the later stages, the algorithm enhances its local search capability and improves solution refinement around the optimal region. To achieve this adaptive behavior, the dynamic perturbation coefficients defined in Equations (37)– (39) are incorporated into the position updating strategy presented

in Equation (40). To further investigate the effectiveness of the proposed approach, four adaptive perturbation strategies namely Exponential β, Linear β, Cosine β, and Quadratic β are integrated into the Modified Seagull Optimization Algorithm (MoSOA). The optimization progress of the algorithm is monitored using convergence curves, which record the best fitness value obtained at the end of each iteration and provide a visual representation of the search performance over time. The performance of MoSOA is evaluated using a set of standard benchmark optimization functions, as presented in Tables 2 and 3. These benchmark functions are grouped into three categories: unimodal, multimodal, and composition functions. The unimodal benchmark functions (F1–F7) contain a single global optimum and are primarily used to evaluate the convergence accuracy and exploitation capability of the optimization algorithms. In contrast, the multimodal benchmark functions (F8–F23) contain multiple local optima and are used to assess the exploration and diversification abilities of the algorithms. These complex multi-peak functions are particularly useful for testing the ability of an optimization algorithm to escape local minima and continue searching for the global optimum.

2.5.4 Hyper parameter tuning using Modified Seagull Optimization Algorithm

The MoSOA methodology has resulted a fitness function (FF) to achieve improved prediction accomplishment. It determines positive values that denotes the dominant achievement of the candidate solutions. Here, the mitigated regression rate of error is the FF, as illustrated in Equation (21). Convergence was tracked by monitoring the stabilization of the fitness values between iterations. For reproducibility, random terms during the optimization procedure were managed through fixed random seeds. The predictive generalization capability and physics-feasibility compliance of PIStGNN, are strongly governed by hyperparameter configuration within a high-dimensional, nonconvex optimization manifold. To overcome these limitations, this work formulates hyperparameter selection as a stochastic population-based optimization problem and employs a Modified Seagull Optimization Algorithm (MoSOA) to identify optimal configurations that minimize a physics-informed learning objective while preserving AC Optimal Power Flow (ACOPF) feasibility. Each search agent (seagull) encodes a candidate hyperparameter vector [d_i, h_i,e_i, η_i,ϱ_i]. where d_i denotes spatiotemporal convolutional depth, h_i latent feature dimensionality, e_i embedding dimension, η_i learning rate, and ϱ_i regularization intensity. The pseudocode is described as follows:

Algorithm 1: MoSOA-PIStGNN

Initialize optimization algorithm parameter u, v, number of population (np), upper bound, lower bound, dimensional space D, maximum number of iterations T,

1. Randomly generate hyperparameter matrix and memory matrix;
2. Load ACOPF data (training input and target)
3. Start loop - 1 for np
4. hyperparameter vector [d_i, h_i,e_i, η_i,ϱ_i] $i = 1,2,\ldots,np$
5. Start loop -2 for number of samples
6. Predict output with PISTGNN architecture
7. End loop -2
8. Evaluate fitness function (29)
9. End loop -1
10. Calculate best fitness function and store hyperparameter values correspondingly
11. Start loop-3 for maximum number of iterations
12. Start loop-4 for number of populations
13. Calculate all variables (46)-(62) and update the positions (63)
14. End loop-4
15. Bound hyperparameter value if it exceeds the range
16. % Calculate new fitness value (29)
17. Start loop-5 for np
18. hyperparameter vector [d_i, h_i,e_i, η_i,ϱ_i] $i = 1,2,\ldots,np$

19. Start loop -6 for number of samples
20. Predict output with PISTGNN architecture
21. End loop -6
22. Evaluate fitness function
23. End loop -5
24. Calculate new best fitness function and store hyperparameter values correspondingly
25. Start loop-7 for np
26. Compare new fitness function with old fitness function and update it in old fitness variable
27. End loop-7
28. Evaluate best fitness value after comparison and save weight values
29. End loop-3
30. Select final hyperparameter values whose fitness function is minimum.

2.5.5 Implementation Details

The is based on Pandapower package which has its own data format stored in pandas dataframe. It is characterized by accelerated computations and less time to transform and extract results. It is also possible to export and import input data to interact with other packages. Although it has the additional time cost of converting the model to and from BBM (Building Block Model). The proposed solvers are to be tested in the following manner. For various IEEE test cases of different sizes, the computation time and the number of iterations in the Newton Raphson method will be measured. An error threshold of 10^{-4} is set as the target accuracy, at which point the iterative algorithm will be halted. The number of iterations indicates the convergence efficiency of different implementations of this method, while the computation time reflects the practical applicability of the solvers. Different warm-start point methods accelerate the optimization process by reducing the iterations. In this experiment, the iterations of the ACOPF solver using the different warm-start point methods are compared. The proposed methodology employs graph convolutional networks to capture the spatial topology of power systems and recurrent architectures (LSTM/GRU) to model temporal dynamics driven by renewable generation variability and load fluctuations. Physical laws, including Kirchhoff's current and voltage laws, are embedded into the training process through customized loss functions to ensure feasibility and interpretability. MoSOA is utilized for adaptive hyperparameter optimization, enhancing convergence stability and model robustness. The trained PIStGNN model is then used to generate initial voltage states for warm-starting the Newton–Raphson solver, thereby improving computational efficiency. The framework is validated using datasets generated from Pandapower simulations on IEEE 33-bus, 57-bus, and 118-bus systems under varying operating conditions and renewable penetration levels. The proposed warm-start approach is evaluated against two conventional initialization methods: the DCOPF-based initialization and the flat start approach, where voltage magnitudes are initialized to 1.0 p.u. and both voltage angles and generation values are set to zero. The comparison is based on the assumption that the computational complexity of each Newton–Raphson iteration is approximately constant. As a result, a lower number of iterations generally corresponds to shorter overall computation time. The experiments were conducted to assess both convergence efficiency and computational performance. Initial tests were performed on a system equipped with an Intel Core i9-10980HK processor and 32 GB of DDR4 RAM. To further evaluate the robustness and stability of the proposed method under more demanding conditions, additional experiments were carried out using the large-scale PEGASE 1354-bus test system. In this scenario, the active and reactive power demands of all loads were increased incrementally by 1% at each step. The computation time and the number of Newton–Raphson iterations were recorded continuously until the solvers could no longer converge within the specified iteration limit. These large-scale stability tests were conducted on a computer with an Intel Core i5-8500H processor and 16 GB of DDR4 RAM.

3. Results and Discussion

3.1 Comparison of different perturbation factors in MoSOA

Figures 2-4 compare the convergence behavior of the four perturbation schedules cosine, exponential, linear, and quadratic over 30 independent runs on benchmark functions (F1)–(F23). The results show that all four strategies are highly competitive on unimodal functions, but clearer differences emerge on multimodal and fixed-dimension problems. Among them, the exponential perturbation is generally the most aggressive and often reaches the best final fitness on multimodal functions. While cosine is competitive and superior on some multimodal cases. In contrast, linear and quadratic tend to be more conservative, sometimes yielding slower convergence or higher final residuals. For the unimodal functions (F1)–(F7), all perturbation strategies exhibit very fast and stable convergence. As shown in Figure 2, F1, the fitness decreases from about 2.0×10^4 to 0 within 8–10 iterations and remain constant afterward. A similar pattern is observed on F2, where the fitness drops from around 180 to almost 0 in fewer than 15 iterations. On F3, the exponential perturbation attains the lowest final value at 10^{-10}, followed by quadratic near 10^{-9}. While linear and cosine remain higher around 10^{-8} to 10^{-9}. On F4, the exponential perturbation again converges fastest and reaches about 10^{-7}, whereas quadratic reaches 2×10^{-7}, linear about 5×10^{-7}, and cosine 10^{-6}. On F5, all methods decrease rapidly from about 10^7 to below 10^1 within roughly 20–25 iterations. On F6, the fitness falls from around 2.0×10^4 to 0 in fewer than 10 iterations for all perturbation functions. On F7, the differences are small but visible: linear achieves the lowest final fitness at about 6×10^{-4}, cosine follows near 9×10^{-4}, while quadratic and exponential settle slightly higher around 1.5×10^{-3}. These results indicate that unimodal functions are relatively insensitive to perturbation design. Although exponential often provides slightly faster refinement on the more precision-sensitive cases. For the multimodal functions (F8)–(F13), the impact of perturbation design becomes more pronounced. On F8, the quadratic perturbation achieves the best final value at -3.7×10^3, followed by exponential near -3.5×10^3, cosine around -3.3×10^3, and linear near -3.1×10^3. On F9, the cosine and exponential schedules both converge to nearly 0 by around 35–40 iterations, whereas linear remains above 0 and quadratic stagnates much higher at about 15. On F10, exponential is best, reaching 10^{-11}, followed by quadratic near 10^{-10}, linear around 10^{-9}, and cosine near 10^{-8}. On F11, all strategies again collapse from about 180 to almost 0 within 15 iterations, showing little separation. On F12, cosine performs best, converging to around 10^{-2}, while linear reaches 2×10^{-2} and quadratic and exponential settle closer to 5×10^{-2}. A similar pattern appears on F13, where cosine attains the lowest final value near 10^{-2}, outperforming linear, quadratic, and exponential, which remain around 10^{-1} to 3×10^{-1}. These results suggest that multimodal problems are more sensitive to perturbation function. The exponential excels on some landscapes requiring stronger exploitation after exploration, and cosine proving effective on functions that benefit from adaptation. For the fixed-dimension functions (F14)–(F23), the four perturbation functions remain close on several benchmarks, but larger gaps emerge on the later functions. On F14, exponential achieves the best final fitness near 1.0, while the other schedules remain around 2.0. On F15, cosine performs best at 5×10^{-4}, followed by linear near 7×10^{-4}, exponential around 8×10^{-4}, and quadratic near 1.3×10^{-3}. On F16, all functions converge to about -1.03 within fewer than 10 iterations. On F17, all four methods also overlap, ending around 0.40. On F18, all perturbation function converges to 3.0, indicating negligible dependence on the chosen schedule. On F19, all strategies stabilize near -3.86, again showing almost identical behavior. More pronounced differences appear on F20, where exponential reaches -3.12, outperforming quadratic near -3.10, linear around -3.07, and cosine near -3.05. On F21, the advantage of exponential becomes clearer: it converges to -10.1, while linear and quadratic settle near -9.5 to -9.8, and cosine remains much higher around -5.0. On F22, exponential again attains the lowest final value at approximately -10.2, marginally better than quadratic and linear, while cosine remains slightly worse near -9.9. On F23, exponential reaches about -5.12, narrowly outperforming quadratic and cosine, while linear remains slightly higher near -5.0. These findings indicate that the fixed-dimension group is mostly

insensitive to perturbation design on easier problems, but exponential becomes increasingly advantageous on more nonlinear landscapes such as F21–F23.

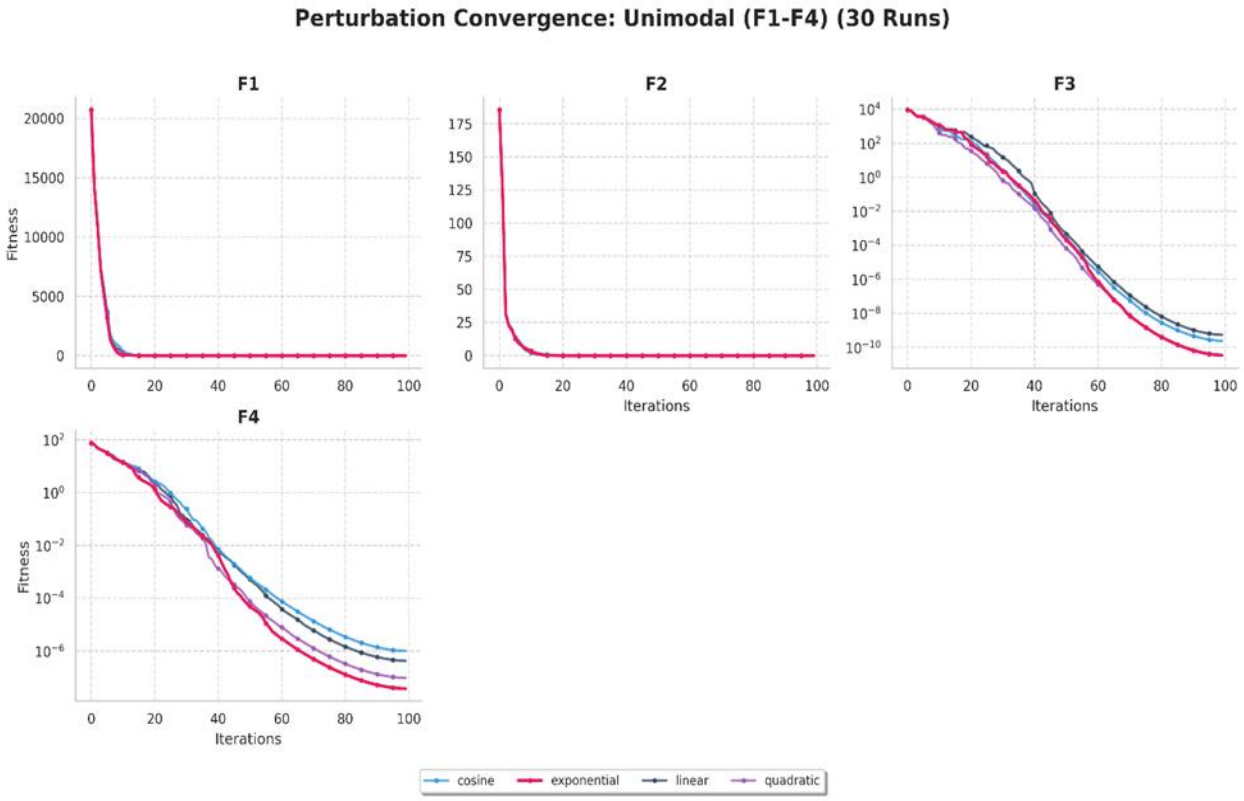


Figure 2 Convergence analysis of perturbation strategies on unimodal benchmark functions

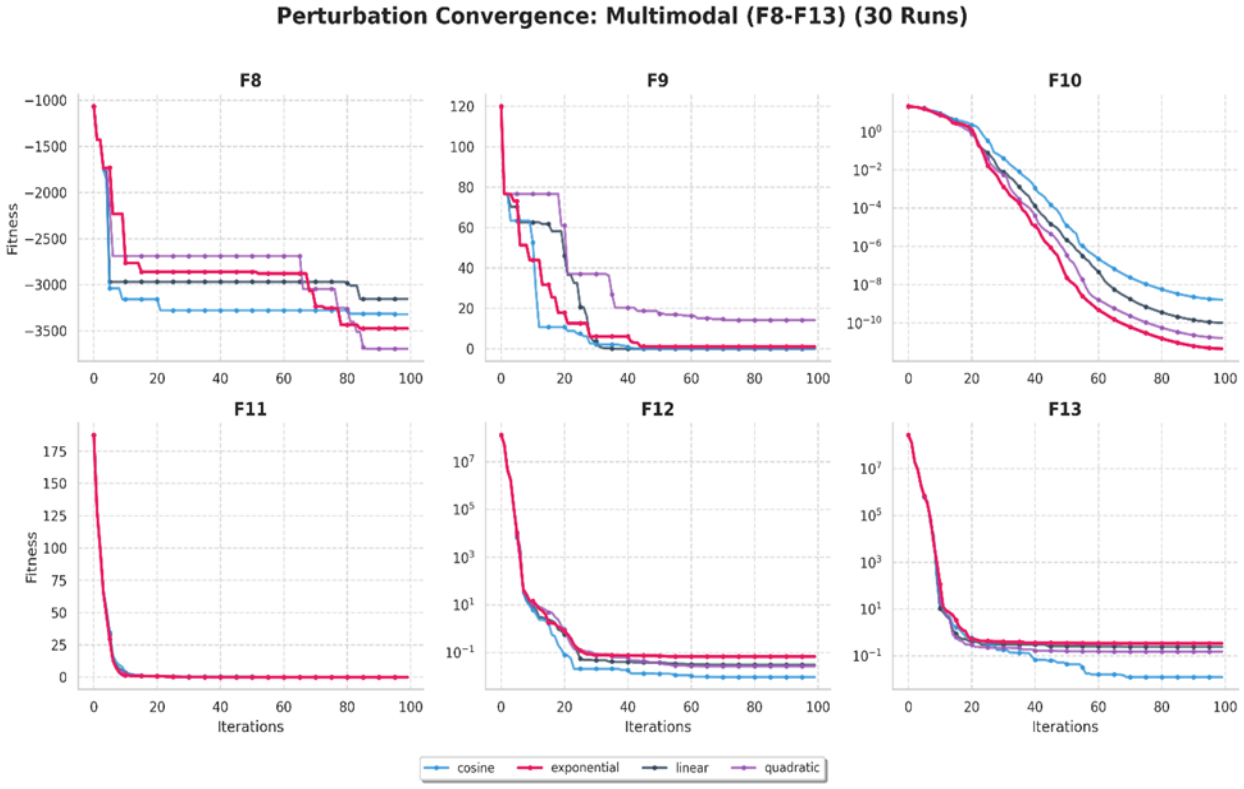


Figure 3 Convergence analysis of perturbation strategies on unimodal benchmark functions

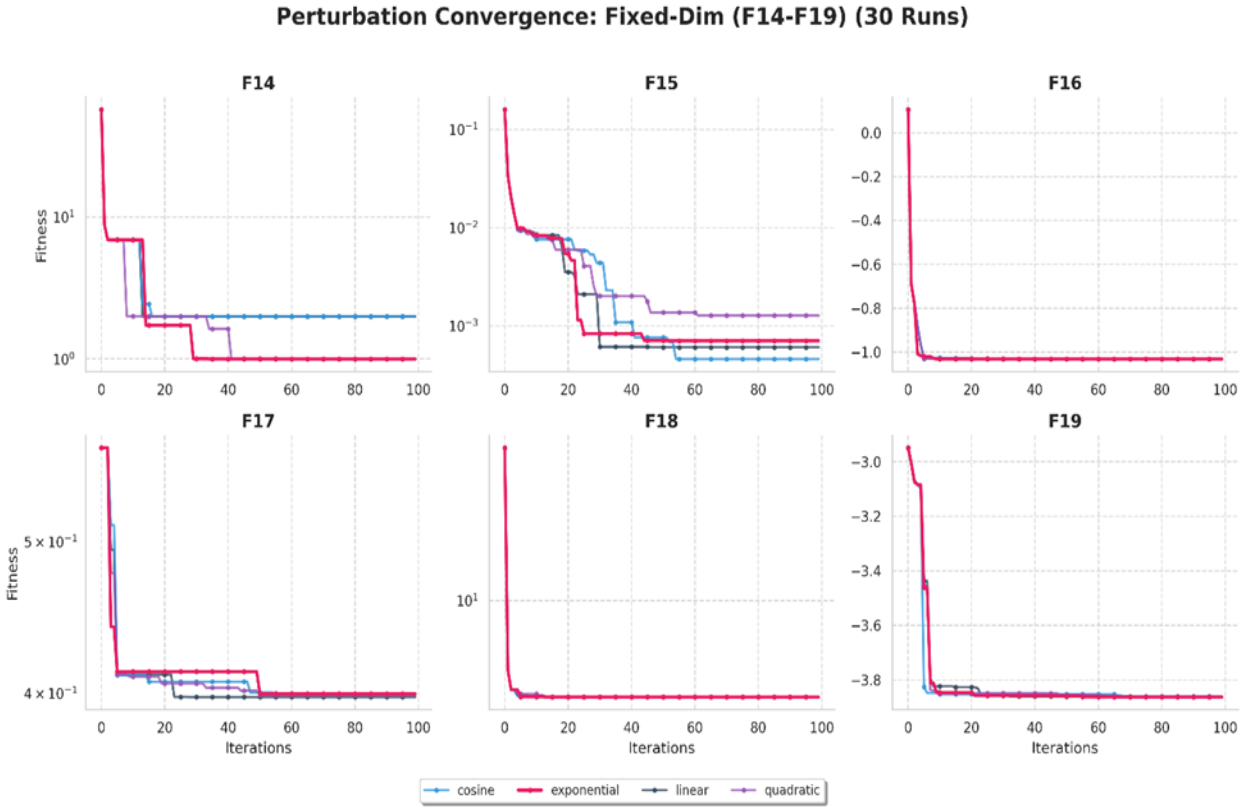


Figure 4 Convergence analysis of perturbation strategies on fixed dimension benchmark functions

3.2 Comparison of MoSOA with Standard Optimization Algorithms

MoSOA performance on the unimodal benchmark functions in Figure 5 shows positive results. It achieves theoretical optima (e.g., F1: 1.45923E-80, F2: 6.30126E-47) with the lowest STDs (e.g., F1: 6.50593E-76), indicating strong global search and stability. Simultaneously, it outperforms all competitors in F1–F5 and F7 and provide the same results ISOA in F6. GA shows poor performance with high means and STDs (e.g., F1 Mean: 1761.053352, STD: 898.4914964), indicating instability. GWO is a strong contender but slightly behind MoSOA (e.g., F1 Mean: 7.6363E-59). PSO is competitive in some cases (e.g., F1 Best: 5.66075E-83) but inconsistent (high STDs). TSA, SOA and ISOA vary in performance, with ISOA closest to MoSOA (e.g., F1 Mean: 6.27755E-35). In summary MoSOA's superiority in unimodal functions highlights its robust global search capability, especially for single-peak problems. In Table 5.2, MoSOA outperforms in F8 (Mean: -6052.43781), F9 (Mean: 10.17984688), F10 (Mean: 6.2883E-15), F11 (Mean: 0.011322154), F12 (Mean: 0.30491761), and F13 (Mean: 1.885161927). But slightly weaker in F9 compared to GWO (Mean: 0.573402645) and in F11 STD compared to GWO. GA consistently poor (e.g., F8 Mean: -4589.698054). GSA is weak across multimodal functions (e.g., F8 Mean: -2509.98029). GWO is competitive in F9 and F11, but less consistent than MoSOA. PSO shows moderate performance, struggles with high STDs (e.g., F9 STD: 15.26444041). TSA/SOA underperform compared to MoSOA and ISOA. ISOA is strong but slightly behind MoSOA (e.g., F8 Mean: -6092.79164). MoSOA excels in multimodal functions, confirming superiority in handling of complex landscapes with multiple optima. Though GWO occasionally challenges it in specific metrics. The MoSOA performed competitive but not always the best. It excels in F21 (Mean: -9.53469252), F22 (Mean: -9.54786511), and F23 (Mean: -9.20154336). Ties with others in F14–F19 but has higher variability in some cases (e.g., F18 STD: 16.03390776). GA and GSA are generally weaker (e.g., F14 Mean: 7.106688339 vs. MoSOA: 7.400445213). GWO and PSO are strong in F14–F19, often matching

theoretical optima (e.g., F16 Mean: -1.03162845). TSA, SOA and ISOA vary, with SOA and ISOA occasionally competitive (e.g., F17 Mean: 0.676485996 for SOA). MoSOA performs well in fixed-dimension multimodal functions but faces stiffer competition, suggesting its strengths are more pronounced in higher-dimensional, variable problems. MoSOA demonstrated the best overall performance, with consistent optima in unimodal functions, strong multimodal performance, and competitive fixed-dimension results. Low STDs indicate high stability. ISOA: Closest competitor, often second-best, confirming the value of SOA improvements. GWO: Strong alternative, particularly in multimodal functions. Others: GA, GSA, PSO, TSA, and SOA lag significantly in most cases.

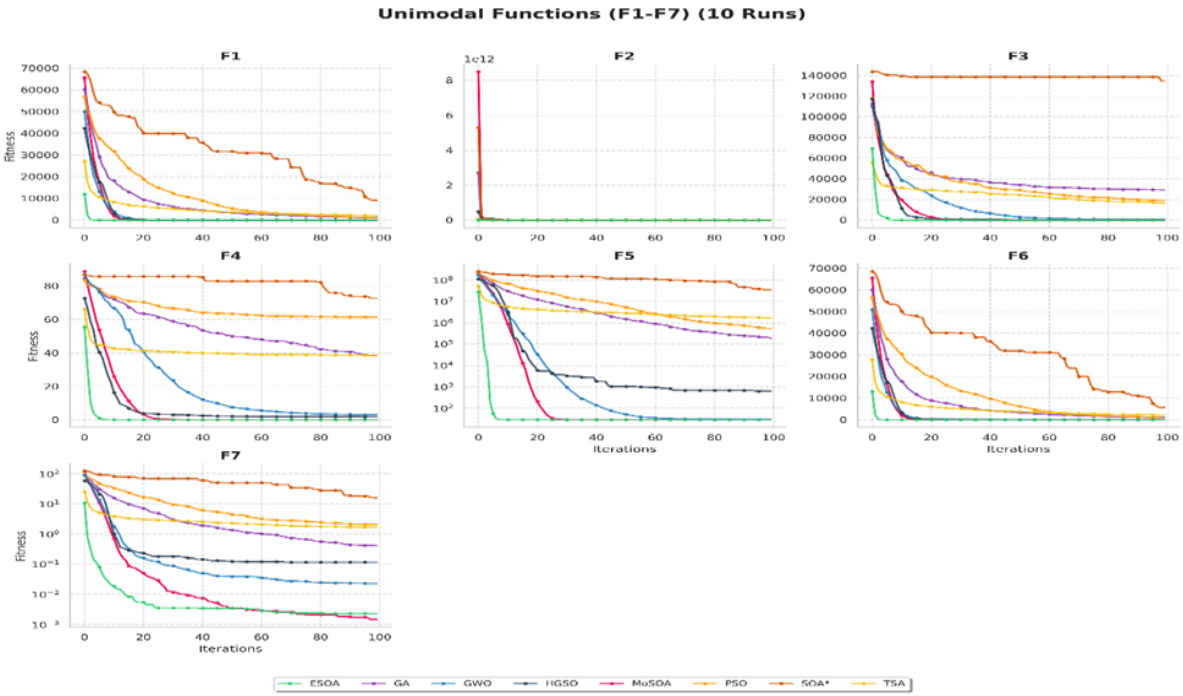


Figure 5 Comparison of MoSOA with standard optimization algorithms unimodal functions (F1)–(F7)

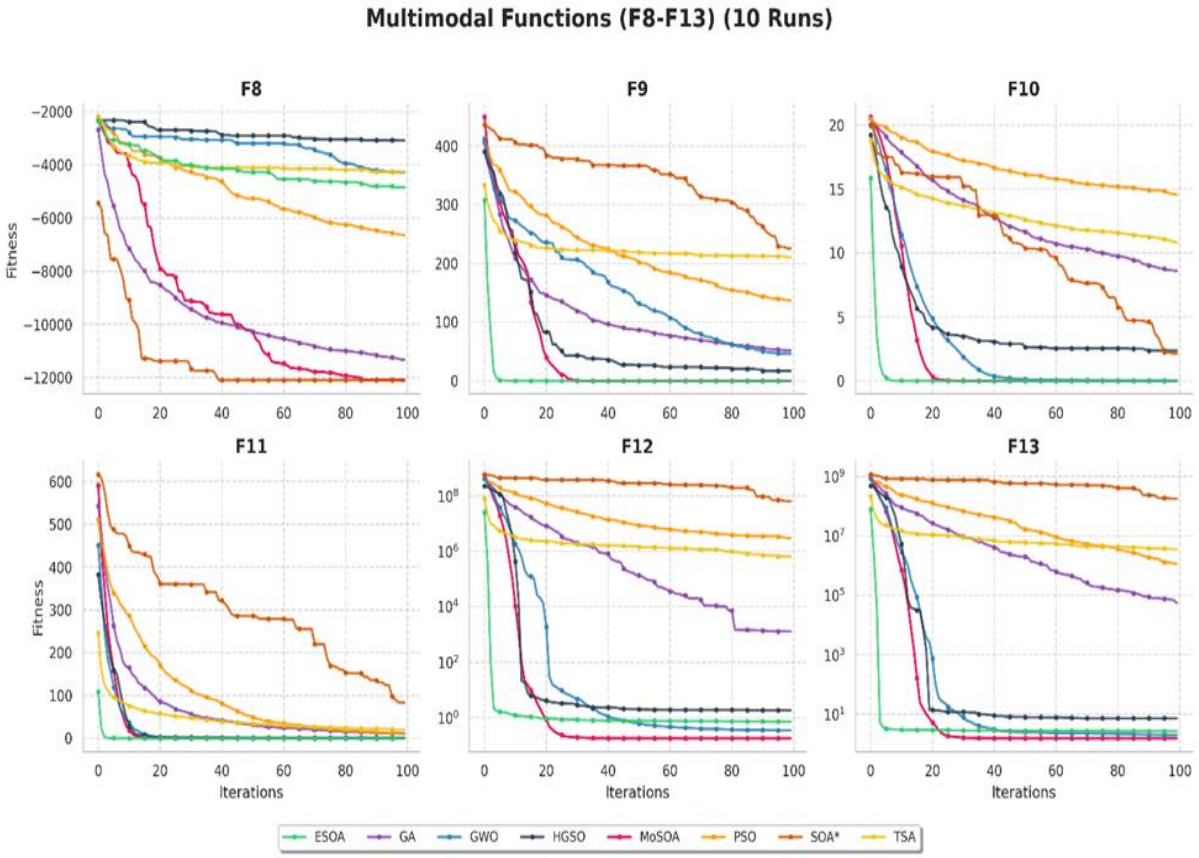


Figure 6 Comparison of MoSOA with standard optimization algorithms unimodal functions (F8)–(F13)

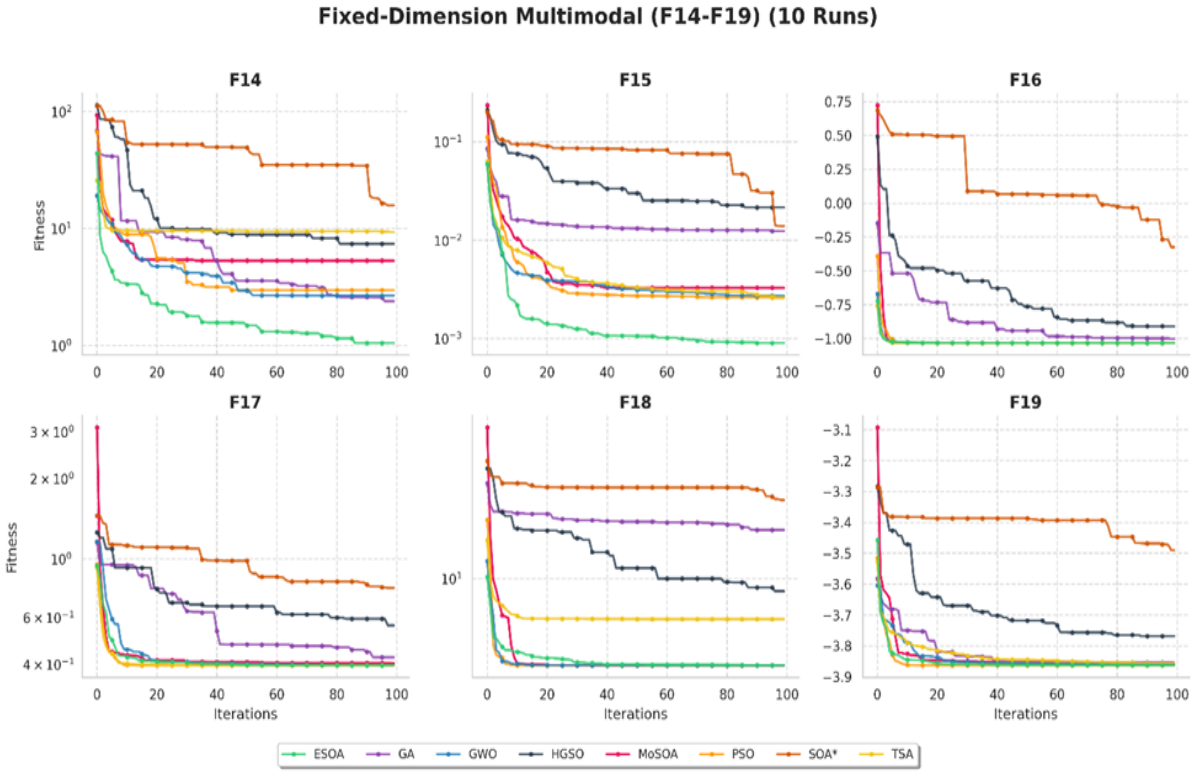


Figure 7 Comparison of MoSOA with standard optimization algorithms Fixed Dimension Multimodal functions (F14)–(F19)

3.3 Analysis of hyperparameter optimization results on the mathematical benchmark

Figures 8-10 compares MoSOA, TPE (Optuna), Random Search, and SOA on the mathematical benchmark in terms of final validation loss, time efficiency, and sample efficiency. The bar chart shows that MoSOA achieves the best final result with a validation loss of 8.31, outperforming the other methods. Hence, relative to MoSOA, TPE is worse by only 0.04 loss units, whereas Random Search and SOA are worse by 3.05 and 4.29 loss units, respectively. The time-based convergence results further highlight the efficiency of MoSOA. In Figure 9, MoSOA reaches a loss below 20 within 0.2 s, and approaches its final value of 8.3 in less than 1 s. By comparison, TPE requires the full-time horizon of about 30 s to converge to its best value, remaining around 17–18 for a large portion of the search before gradually improving. This indicates that MoSOA reaches a better final solution using 30× less than TPE. The gap is even more pronounced against the weaker optimizers: SOA drops quickly at first but plateaus around 12.6, while Random Search stagnates near 11.36. Both failing to approach the lower-loss regime achieved by MoSOA. Figure 10 shows similarly strong advantage. MoSOA reduces the validation loss to below 10 after 80–100 trials. It then stabilizes near 8.3 by 500 trials. TPE reaches a comparable final loss, but only after 1000 trials. In contrast, Random Search remains around 11.36 even after 1000 trials, while SOA converges more slowly and levels off near 12.6 after several hundred trials. Thus, MoSOA not only finds the best solution overall, but does so with markedly fewer function evaluations. The numerical trends also indicate differences in optimizer behavior. MoSOA improves from an initial loss near 29–30 to 8.31, corresponding to an absolute reduction of about 21 loss units. TPE decreases from roughly 30 to 8.35, achieving a similar overall reduction but over a much longer time and larger number of trials. Random Search starts above 40 and drops to 11.5, while SOA decreases from about 33 to 12.60. It evident that MoSOA combines the lowest final loss, the fastest convergence in time, and the highest sample efficiency. Overall, these results show that MoSOA provides the best trade-off between accuracy and efficiency on the mathematical benchmark. Its final loss advantage over TPE is modest but measurable. More importantly, MoSOA reaches this solution in under 1 s and within roughly 100 trials for near-optimal performance, whereas TPE requires about 30 s and close to 1000 trials to reach a similar region. These numerical findings strongly support the effectiveness of MoSOA for hyperparameter optimization in settings where both optimization quality and computational budget are important.

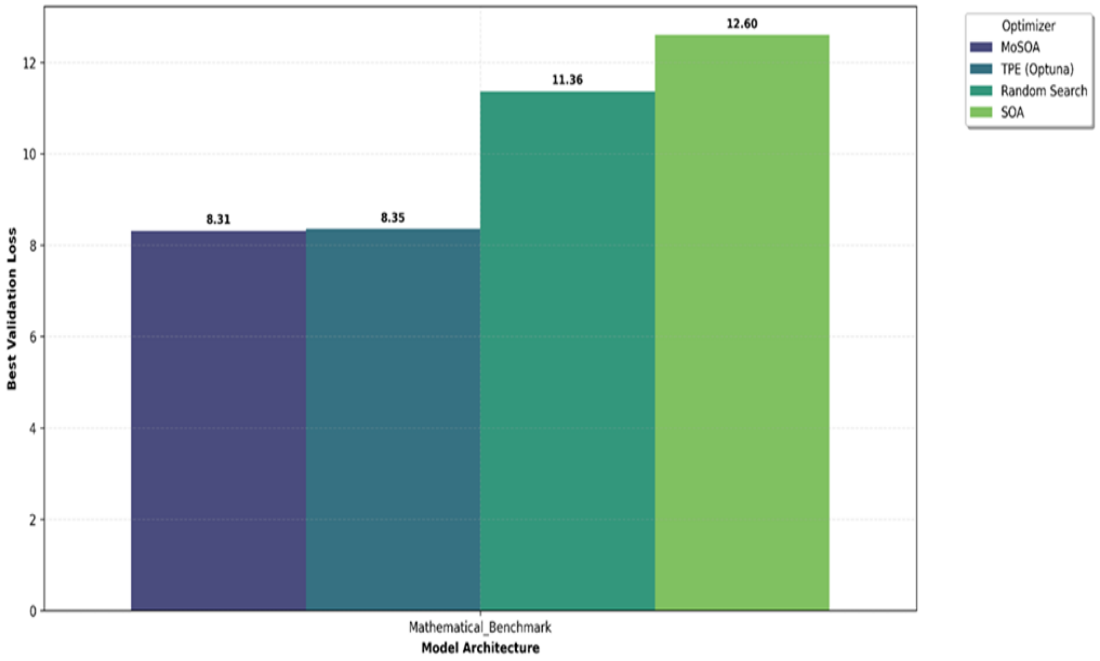


Figure 8 Analysis of hyperparameter optimizer performance on the mathematical benchmark

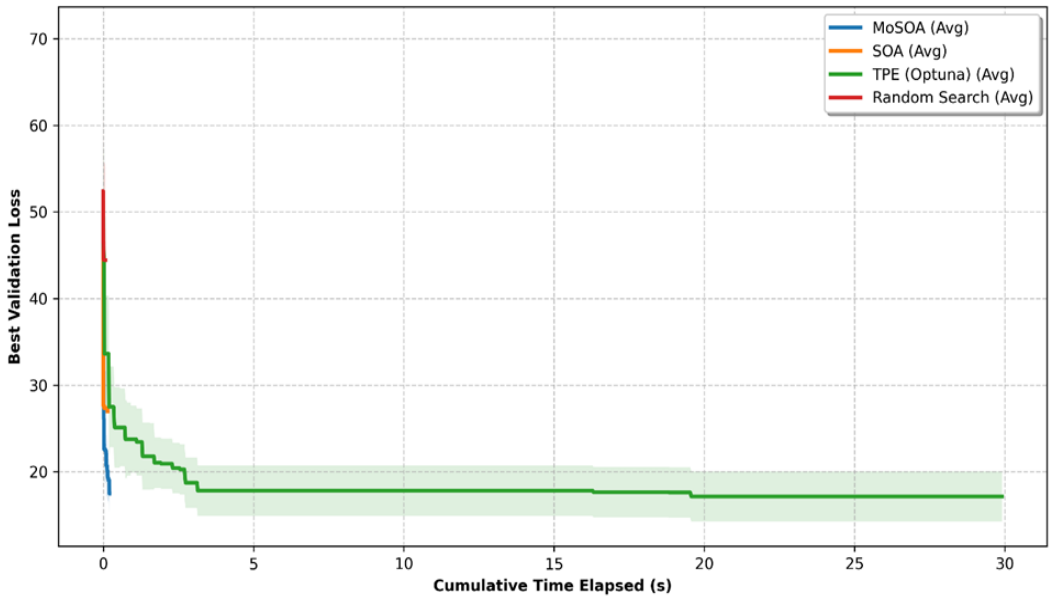


Figure 9 Analysis of hyperparameter optimizer computational efficiency on benchmark function

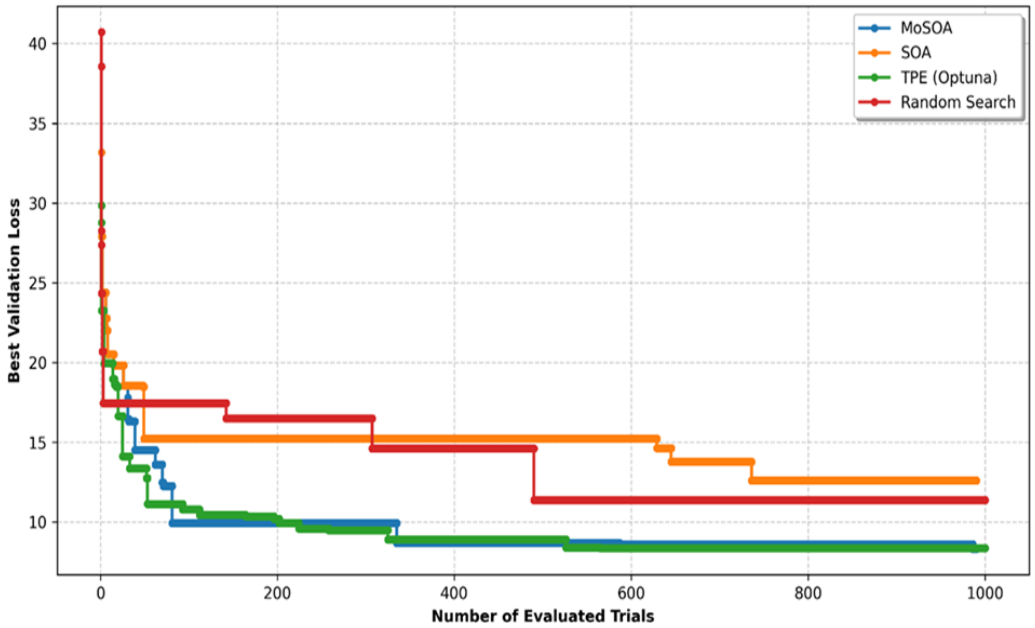


Figure 10 Analysis of Hyperparameter Optimization Results on the Mathematical Benchmark

3.4 Warm-starting Newton Raphson solver for ACOPF prediction using physics-informed spatiotemporal graph neural network.

Tables 4-6 compare DC and flat start with PIStGNN warm-start strategies for IEEE 33-, 57-, and 118-bus systems. Table 4 all methods converge in nearly three iterations on IEEE 33-bus case. The voltage magnitude MAE remains almost identical across methods. This suggests that the 33-bus system is easy for the Newton–Raphson solver. The initial condition has limited influence on convergence accuracy. However, some PIStGNN initializations, particularly PIGCLSTM and DynamicGCN, reduce computation time compared with DC and flat initialization. In Table 5 the DC initialization remains the most iteration-efficient method. Several physics-informed graph models, especially PIGCGRU, PIGCLSTM, PIGCN, and PIResnetGCLSTM, achieve competitive computation times. With voltage MAE values very close to the DC

and flat baselines. The dynamicGCN and standardGCN require more iterations, above 5 on average. Table 6 provides the strongest evidence of scalability differences. Flat start requires 4 iterations, while DC initialization remains stable at 3 iterations. Among the learning-based approaches, PIGCN achieves the fastest average computation time in its comparison block and converges in about 3.29 iterations. This is close to the DC baseline and better than flat start. PIGCLSTM also performs well, requiring only 3.5 iterations with lower computation time than both DC and flat cases. By contrast, StandardGCN and DynamicGCN require more iterations, 6.6 and 5.58 respectively. It indicates that purely data-driven graph embeddings may not always provide a suitable initialization for nonlinear ACOPF refinement. However, physics-informed training predicts voltage states, but also support numerical convergence. Although the proposed PIStGNN-based warm-start methods require slightly higher Newton–Raphson iteration counts compared to conventional DC initialization, the overall computational time is reduced. This indicates that iteration count alone is not a sufficient measure of computational efficiency. The PIStGNN-generated initial conditions provide a numerically smoother and better-conditioned starting point. This leads to reduced mismatch magnitudes and more stable Jacobian updates during the iterative process. Consequently, the computational cost per iteration is reduced. This results in faster convergence despite the marginal increase in iteration count. A key observation across the three systems is that voltage magnitude MAE changes only slightly between initialization methods after Newton–Raphson convergence. This is expected because the solver refines each initialization toward an AC-consistent solution. Therefore, the main performance advantage of warm-starting is not necessarily a lower final MAE, but rather reduced computation time, and improved convergence behavior. The results suggest that PIStGNNs provide an initialization that allows Newton–Raphson to converge efficiently while preserving physical correctness. This observation highlights the importance of considering both iteration complexity and numerical conditioning when evaluating warm-start strategies for ACOPF solvers

Table 4 Warm-starting Newton Raphson solver for ACOPF prediction on IEEE 33

Initialization Method	Time (ms)	Iterations	MAE (VM)
DC	37.2427028	2.9	0.0126317882
FLAT	36.8320545	2.88	0.0126317112
DynamicGCN	30.3906618	3	0.0126318532
DC	44.3576984	2.9	0.0126317882
FLAT	39.5773426	2.88	0.0126317112
PIGCGRU	36.4684445	3	0.0126318893
DC	33.9315182	2.9	0.0126317882
FLAT	30.1921833	2.88	0.0126317112
PIGCLSTM	27.8152471	3	0.0126318889
DC	40.2853444	2.9	0.0126317882
FLAT	37.0137716	2.88	0.0126317112
PIGCN	32.7331652	2.92	0.0126318358
DC	44.2084863	2.9	0.0126317882
FLAT	39.5461623	2.88	0.0126317112
PIResnetGCGRU	36.3585597	3	0.0126318885
DC	44.2565157	2.9	0.0126317882
FLAT	39.6743123	2.88	0.0126317112
PIResnetGCLSTM	36.3576012	3	0.0126318892
DC	42.6459513	2.9	0.0126317882
FLAT	49.4777718	2.88	0.0126317112
StandardGCN	35.0323992	3	0.0126318601

Table 5 Warm-starting Newton Raphson solver for ACOPF prediction on IEEE 57

Initialization Method	Time (ms)	Iterations	MAE (VM)
DC	49.5865169	3	0.044657720
FLAT	55.2998164	3.3333333	0.044657921
DynamicGCN	52.5485420	5.6190476	0.044867979
DC	64.2662993	3	0.044657720
FLAT	61.2981568	3.3333333	0.044657921
PIGCGRU	56.3226313	3.4583333	0.044658312
DC	50.0939049	3	0.044657720
FLAT	47.9321250	3.3333333	0.044657921
PIGCLSTM	44.9495497	3.8333333	0.044658507
DC	49.5716059	3	0.044657720
FLAT	47.3698102	3.3333333	0.044657921
PIGCN	44.4345278	3.7083333	0.044658462
DC	49.6969579	3	0.044657720
FLAT	47.2193653	3.3333333	0.044657921
PIResnetGCGRU	45.0489757	3.7916667	0.044658475
DC	56.7773206	3	0.044657720
FLAT	54.8839958	3.3333333	0.044657921
PIResnetGCLSTM	49.3648073	3.4583333	0.044658076
DC	58.5957070	3	0.044657720
FLAT	55.8890807	3.3333333	0.044657921
StandardGCN	56.5454890	5.8	0.048546682

3.5 Scalability analysis of the proposed Warm-started Newton Raphson solver

Figure 11 presents the comparative ACOPF solve times obtained using FLAT start, DC initialization, and the proposed PIStGNN-based initialization across the IEEE 33-, 57-, and 118-bus systems. Although several machine learning-based ACOPF warm-start approaches have been reported in the literature, including the works of Baker (2019), Tudoras-Miravet et al. (2024), Okhuegbe et al. (2024), and Cao et al. (2023), direct experimental comparison was not conducted in this study due to differences in datasets, and network configurations. Nevertheless, the proposed PIStGNN framework was comprehensively evaluated against conventional initialization techniques, including DC and flat-start initialization, across IEEE 33-, 57-, and 118-bus systems. The results demonstrate that the proposed approach achieves lower computational times than the conventional initialization methods. For the IEEE 33-bus system, the proposed method achieved the lowest solve time of 33.7 ms, compared to 38.9 ms for FLAT initialization and 41.0 ms for DC initialization. This corresponds to a computational time reduction of 13.4% relative to FLAT start and 17.8% relative to DC initialization. The reduction indicates that the PIStGNN-based warm-start provides a more informed initial operating point, thereby accelerating convergence of the Newton–Raphson ACOPF solver. A similar trend is observed for the IEEE 57-bus network. The proposed method recorded a solve time of 49.0 ms, whereas FLAT and DC initializations required 52.9 ms and 54.1 ms, respectively. This represents a runtime improvement of 7.4% over FLAT start and 9.4% over DC initialization. The reduced computational burden suggests that the spatiotemporal graph representations effectively capture both network topology and temporal operating dynamics, leading to improved initialization quality. For the larger IEEE 118-bus system, the proposed method achieved a solve time of 60.3 ms, compared to 64.2 ms for FLAT initialization and 61.1 ms for DC initialization. Although the relative improvement decreases as the network size increases, the proposed framework still provides 6.1% improvement over FLAT start and about 1.3% over DC

initialization. This reduction demonstrates that the PIStGNN warm-start remains computationally beneficial even for larger systems. On average, the proposed method achieved 1.3–18% computational time reduction** compared to conventional initialization techniques. These findings confirm the effectiveness of physics-informed spatiotemporal graph learning in improving ACOPF warm-start quality and accelerating Newton–Raphson convergence performance.

Table 6 Warm-starting Newton Raphson solver for ACOPF prediction on IEEE 118

Initialization Method	Time (ms)	Iterations	MAE (VM)
DC	51.5249355	3	0.0063334
FLAT	53.0479749	4	0.0063334
DynamicGCN	54.7154476	5.5833333	0.0024845
DC	66.6485204	3	0.0063334
FLAT	79.9917564	4	0.0063334
PIGCGRU	67.9206388	5	0.0063334
DC	64.1859893	3	0.0063334
FLAT	65.5587863	4	0.0063334
PIGCLSTM	56.9740362	3.5	0.0063334
DC	53.2595939	3	0.0063334
FLAT	53.50158738	4	0.0063334
PIGCN	44.19419187	3.2916667	0.0063334
DC	66.83849067	3	0.0063334
FLAT	69.14876325	4	0.0063334
PIResnetGCGRU	62.00462829	4	0.0063334
DC	66.71775004	3	0.0063334
FLAT	67.45129646	4	0.0063334
PIResnetGCLSTM	68.97450204	5	0.0063334
DC	59.79743108	3	0.0063334
FLAT	60.40046179	4	0.0063334
StandardGCN	72.19992570	6.6	0.0054637

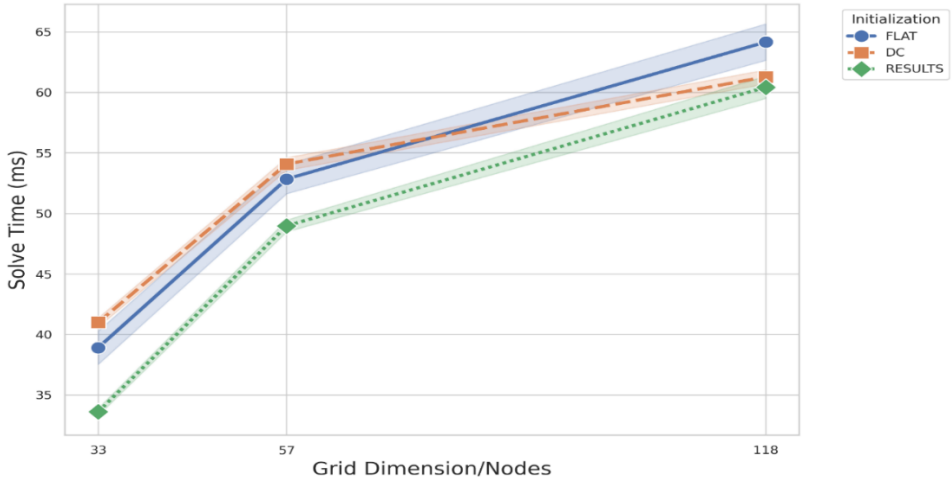


Figure 11 Scalability analysis of the solver computational time

4. Conclusions

This study accelerated the solution of the AC Optimal Power Flow (ACOPF) problem using PIStGNN and the Newton–Raphson (NR) solver. The core contribution lies in warm-starting the iterative NR solver, thereby enhancing convergence efficiency and computational performance. The proposed methodology addresses the challenge of sensitivity to initialization in Newton-based solvers. The traditional approaches such as flat start and DC initialization either require additional preprocessing or fail to generalize under dynamic operating conditions, especially in sustainable grids. the PIStGNN model learns complex nonlinear mappings between system states and optimal solutions while embedding physical laws into the training process. Furthermore, the use of MoSOA for hyperparameter tuning efficiently navigate the high-dimensional, nonconvex optimization space. The simulation results across IEEE 33-bus, 57-bus, and 118-bus systems demonstrate that the proposed PIStGNN achieves reduced computational time and maintains solution accuracy comparable to traditional methods. The results also confirm that PIStGNN warm-starting reduces solver effort without compromising feasibility. Therefore, the model is suitable for real-time power system operation. Despite physics-informed losses improve feasibility, the perfect constraint satisfaction still relies on the downstream NR solver. Future work should improve generalization across unseen grid topologies and deploy the model large-scale smart grids. Future work will focus on implementing standardized benchmarking against state-of-the-art ML-based ACOPF warm-start methods to further validate the comparative performance and generalization capability of the proposed approach.

Declaration of Ethical Standards

As the authors of this study, we declare that he complies with all ethical standards.

Credit Authorship Contribution Statement

A.-M. I. Malori: conceptualization, Validation, Formal analysis, Visualization and methodology

E. A. Frimpong: supervised to ensure technical accuracy of the results

F. E. Boafo: supervised to ensure technical accuracy of the results

E. Twumasi: supervised to ensure technical accuracy of the results

P. Asigri: supervised to ensure technical accuracy of the results

B. M. Adjei: software implementation and experimentation.

Declaration of Competing Interest

The authors declared that they have no conflict of interest.

Funding / Acknowledgements

The authors were supported within the scope of the research by the KNUST Engineering Education Project, Power System Laboratory. The authors declare that generative AI was used only for language editing and not for scientific content generation.

Data Availability

The source code and dataset are publicly available on GitHub repository;
https://github.com/MajidMalori/Optimal-Power-Flow-Prediction-Using-Physics-Informed-Spatiotemporal-Graph-Neural-Network-

References

Ahmad, T., Hamilton, R., Stiasny, J., Chevalier, S., Nellikkath, R., Murzakhanov, I., . . . Papadopoulos, P. (2022). Interpretable Machine Learning for power systems: Establishing Confidence in SHapley Additive ExPlanations. *In Proc. Climate Change AI Workshop ICLR (Tackling Climate Change with Machine Learning).* doi:https://doi.org/10.48550/arXiv.2209.05793

Baker, K. (2019). Learning Warm-Start Points For Ac Optimal Power Flow. *IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP).* Pittsburgh, PA, USA. doi:https://doi.org/10.1109/MLSP.2019.8918690

Bassey, K. E. (December 2023). Hybrid Renewable Energy Systems Modeling. *Engineering Science & Technology Journal, 4*(6), 571-588. doi:10.51594/estj/v4i6.1255

Berizzi, A., Ilea, V., Petrelli, M., Vicario, A., Bovo, C., Carlini, E. M., . . . Zaottini, R. (2021). OPF model with dynamic security constraints: a state of the art review. *AEIT International Annual Conference (AEIT)*. Milan, Italy. doi:https://doi.org/10.23919/AEIT53387.2021.9627047

Cao, Y., Zhao, H., Liang, G., Zhao, J., Liao, H., & Yang, C. (2023). Fast and explainable warm-start point learning for AC Optimal Power Flow using decision tree. *International Journal of Electrical Power and Energy Systems, 153*, 1-9. doi:https://doi.org/10.1016/j.ijepes.2023.109369

Casella, F., & Bachmann, B. (2021). On the choice of initial guesses for the Newton-Raphson algorithm. *Applied Mathematics and Computation, 398*, 1-38. doi:https://doi.org/10.1016/j.amc.2021.125991

Chen, P., Li, H., He, F., & Bian, D. (2024). Multi-strategy improved seagull optimization algorithm and its application in practical engineering. *Engineering Optimization, 56*(12), 1-40. doi:https://doi.org/10.1080/0305215X.2024.2378352

Deihim, A., Apostolopoulou, D., & Alonso, E. (2024). Initial estimate of AC optimal power flow with graph neural networks. *Electric Power Systems Research, 234*, 1-7. doi:https://doi.org/10.1016/j.epsr.2024.110782

Deng, J.-J., & Chiang, H.-D. (2013). Convergence Region of Newton Iterative Power Flow Method: Numerical Studies. *Journal of Applied Mathematics*, 1-12. doi:https://doi.org/10.1155/2013/509496

Dhiman, G., & Kumar, V. (2019). Seagull optimization algorithm: Theory and its applications for large-scale industrial engineering problems. *Knowledge-Based Systems, 165*, 169–196. doi:https://doi.org/10.1016/j.knosys.2018.11.024

Frank, S., & Rebennack, S. (2016). An introduction to optimal power flow: Theory, formulation, and examples. *IIE Transactions, 48*(12), 1172–1197. doi:https://doi.org/10.1080/0740817X.2016.1189626

Gonggui Chen, Tan, T., Xiang, W., Guan, Z., Tan, H., Yu, J., & Long, H. (2023). Solving Environment Economic Power Dispatch Problems by Multi-objective Modified Seagull Optimization Algorithm with Novel Constraint Treatments. *IAENG International Journal of Applied Mathematics, 53*(1), 1-17.

Guha, N., Wang, Z., Wytock, M., & Majumdar, A. (2019). Machine Learning for AC Optimal Power Flow. *arXiv preprint arXiv:1910.08842*. doi:https://doi.org/10.48550/arXiv.1910.08842

Huang, B., & Wang, J. (2023). Applications of Physics-Informed Neural Networks in Power Systems - A Review. *IEEE Transactions on Power Systems,, 38*(1), 572-588. doi:https://doi.org/10.1109/TPWRS.2022.3162473

Khaloie, H., Dolányi , M., Toubeau, J.-F., & Vallée, F. (2025). Review of machine learning techniques for optimal power flow. *Applied Energy, 388*(0306-2619), 125637. doi:https://doi.org/10.1016/j.apenergy.2025.125637

Li, L.-L., Zheng, S.-J., Tseng, M.-L., & Liu, Y.-W. (2021). Performance assessment of combined cooling, heating and power system operation strategy based on multi-objective seagull optimization algorithm. *Energy Conversion and Management, 244*, 114443. doi:https://doi.org/10.1016/j.enconman.2021.114443

Lin, Y., Tang, J., Guo, J., Wu, S., & Li, Z. (2025). Advancing AI-Enabled Techniques in Energy System Modeling: A Review of Data-Driven, Mechanism-Driven, and Hybrid Modeling Approaches. *Energies, 18*, 1-28. doi:https://doi.org/10.3390/en18040845

Oda, E. S., Amr, A. A., Abdelsalam, A. A., & Salem, A. A. (2022). Unit Commitment in Presence of Renewable Energy using Rat and Seagull Optimization Algorithm. *23rd International Middle East Power Systems Conference (MEPCON)*, (pp. 1-8). Cairo, Egypt. doi:https://doi.org/10.1109/MEPCON55441.2022.10021805

Okhuegbe, S. N., Ademola, A. A., & Liu, Y. (2024). A Machine Learning Initializer for Newton-Raphson AC Power Flow Convergence. *IEEE Texas Power and Energy Conference (TPEC)*. College Station, TX, USA. doi:https://doi.org/10.1109/TPEC60005.2024.10472261

Paramguru, J., Barik, S. K., Sahoo, A. K., & Parida, T. (2022). Optimization of Dynamic Economic Dispatch Problem for Micro Grid with Incorporation of Wind Energy by Using Seagull Optimization. *IEEE World Conference on Applied Intelligence and Computing* (pp. 1-5). Sonbhadra, India: IEEE. doi:https://doi.org/10.1109/AIC55036.2022.9848893

Qin, J., Yang, R., & Yu, N. (2025). Physics-Informed Graph Neural Networks for Collaborative Dynamic Reconfiguration and Voltage Regulation in Unbalanced Distribution Systems. *IEEE Transactios on Industry Applications,, 61*(2), 2538-2548. doi:https://doi.org/10.1109/TIA.2025.3529799

Saini, A., & Rahi, O. P. (2024). Optimal power flow approaches for a hybrid system using metaheuristic techniques: a comprehensive review. *International Journal of Ambient Energy, 45*(1), 2345839.

doi:https://doi.org/10.1080/01430750.2024.2345839

Sun, Q., Liu, L., Ma, D., & Zhang, H. (2017). The Initial Guess Estimation Newton Method for Power Flow in Distribution Systems. *IEEE/CAA Journal of Automatica Sinica,, 4*(2), 231-242. doi:https://doi.org/10.1109/JAS.2017.7510514

Tudoras-Miravet, À., Gonzalex-Iakl, E., & Gomis-Bellmunt, O. (2024). Physics-Informed Neural Networks for Power Systems Warm-Start Optimization. *IEEE Access, 12*, 135913-135928. doi:https://doi.org/10.1109/ACCESS.2024.3406471

Vyakaranam, B., Nguyen, Q. H., Nguyen, T. B., Samaan, N. A., & Huang , R. (2021). Automated Tool to Create Chronological AC Power Flow Cases for Large Interconnected Systems. *8*, IEEE open access journal of power and energy. doi:https://doi.org/10.1109/OAJPE.2021.3075659

Wang, B., & Tan, J. (2022). *DC-AC Tool: Fully Automating the Acquisition of the AC Power Flow Solution.* 15013 Denver West Parkway, Golden, CO 80401: National Renewable Energy Laboratory. Retrieved from https://www.nrel.gov/docs/fy22osti/80100.pdf

Wei, H., Xu , K., & Zhang, J. (2022,). Enhanced Seagull Optimization Algorithm for Photovoltaic Cell Parameter Estimating. *Proceedings of the 41st Chinese Control Conference*, (pp. 1-6). Hefei, China. doi:https://doi.org/10.23919/CCC55666.2022.9901592