Electric Power Systems Research 234 (2024) 110782

Contents lists available at ScienceDirect

Electric Power Systems Research

journal homepage: www.elsevier.com/locate/epsr

Initial estimate of AC optimal power flow with graph neural networks

Azad Deihim [a,*], Dimitra Apostolopoulou [a], Eduardo Alonso [b]

[a] Department of Engineering, City, University of London, London EC1V 0HB, UK
[b] Department of Computer Science, City, University of London, London EC1V 0HB, UK

ARTICLE INFO

Keywords:
AC optimal power flow
Graph neural networks
Initial estimate
Proximal policy approximation

ABSTRACT

Optimal power flow (OPF) is a crucial task in power system management and control; accurate and time-efficient solutions for OPF are necessary to ensure cost-efficient and reliable power system operation. We introduce a novel solution to solving alternating current OPF (ACOPF), a nonlinear and nonconvex optimization problem, by combining the speed of a trained deep learning model with the accuracy of iterative solvers. The proposed framework uses a graph neural network (GNN) to exploit the graph structure of a power system in conjunction with proximal policy optimization, a deep reinforcement learning algorithm, to compute initial guesses for an interior point solver (IPS), providing a warm start, allowing the solver to converge in fewer iterations. Existing literature that explores warm start ACOPF solutions using machine learning choose to compute initial guesses that are trained to be feasible and cost-minimizing. Our approach trains the GNN-based reinforcement learning agent to produce an output that minimizes IPS convergence time by designing a reward function that is a function of the IPS convergence time. We evaluate the proposed framework using IEEE test case environments, using PyPower's IPS-based ACOPF solver and a GNN-based framework that computes ACOPF solutions directly as baselines, demonstrating significantly improved convergence times.

1. Introduction

Optimal power flow (OPF) is a fundamental task in power system management that aims to minimize power generation costs under the physical constraints of the power system. OPF is an NP-Hard problem in its original form due to the nonlinear and nonconvex properties of alternating current OPF (ACOPF). In the last two decades, finding direct solutions for ACOPF has become a popular area of research [1]. Proposed solutions often fall short because they cannot consistently produce feasible solutions, achieve global optimum, or address space–time complexity[1] issues related to solving a nonconvex optimization problem. The alternative to solving ACOPF is to apply a direct current (DC) approximation using a relaxation that negates the reactive component(s) of the power system — thus making the optimization problem linear and convex. DCOPF is a common approach for solving OPF due to the lack of computational complexity associated with solving it. Nonetheless, the approximation of OPF's true form will incur significant monetary losses due to suboptimal solutions obtained using DCOPF [2].

Deep learning approaches to solving ACOPF have been well explored (see, e.g., [3–6]). While deep learning has been shown to compute feasible solutions quickly, they are consistently far from global optimum relative to other methods, such as iterative solving methods. Iterative solvers, such as gradient descent, Newton–Raphson's method, or quasi-Newton methods, are numerical methods that use an initial value to generate a sequence of increasingly accurate approximate solutions for a specific class of problems. In these methods, the nth approximation is derived from the previous approximations, gradually refining the solution until it reaches a global optimum. Due to neural networks' inherent functionality for generalization, it would be nearly impossible to generate a truly optimal output. The only means of consistently producing optimal solutions would be to create an architecture designed to overfit and then train it on every possible combination of inputs, which is infeasible for a continuous, thus infinitely large, input space. Alternatively, while global optimality is not necessarily guaranteed every time, iterative solvers alone are generally capable of consistently producing optimal ACOPF solutions that are better solutions than other researched methods, such as deep learning, but this comes at the cost of computation time; convergence to the optimum can often not be done in real-time, especially in larger power systems. This problem can be mitigated if the iterative solvers are informed with an accurate initial guess or warm-start. Providing a warm-start can significantly improve the convergence rate and reduce the number of iterations required to converge [7].

This paper proposes a novel time-efficient solution to ACOPF using deep reinforcement learning in conjunction with iterative solving

* Corresponding author.
E-mail addresses: azad.deihim@city.ac.uk (A. Deihim), dimitra.apostolopoulou@city.ac.uk (D. Apostolopoulou), e.alonso@city.ac.uk (E. Alonso).

[1] Space–time complexity refers to an algorithm's efficiency in computation time and required memory.

https://doi.org/10.1016/j.epsr.2024.110782
Received 1 October 2023; Received in revised form 15 March 2024; Accepted 17 June 2024
Available online 25 June 2024

methods, specifically, the interior point solver (IPS). We do not train the deep reinforcement learning agent to produce a solution to ACOPF that is feasible or cost-minimizing; instead, we train the agent to compute an initial guess that minimizes the time it would take for the IPS to find a solution. Due to the nonconvex properties of ACOPF, an initial guess that is nearer to the global optimum in terms of costs and feasibility may be very far in proximity to the global optimum. As a result, training an agent to generate initial guesses that prioritize feasibility and cost minimization may actually lead to longer IPS convergence times.

Training the agent to make initial guesses that minimize IPS computation time requires a feedback loop to inform the loss function of the quality of the initial guess based on the amount of time it takes for the IPS to solve the ACOPF. This approach should employ reinforcement learning, as it would not be a practical way to generate a predefined dataset for traditional machine learning approaches.

Our proposed framework uses proximal policy optimization (PPO), a type of deep reinforcement learning algorithm, to train a neural network to compute an initial guess of the ACOPF solution, allowing IPS to solve it in significantly fewer iterations, thus decreasing the computation time. Reinforcement learning agents learn via trial and error as they interact with their environment, take actions, and observe the rewards earned from making particular actions. PPO is a simple actor–critic reinforcement learning algorithm that seeks to find an optimal policy (a function that maps observations to actions) rather than assigning values to state–action pairs [8]. We couple this with a graph neural network (GNN) to act as the agent's learning mechanism — exploiting the graph structure of the power grid. GNNs are a specialized type of neural networks that are well-suited to process graph-structured data. By analyzing and utilizing the patterns and relationships within the graph, GNNs can provide more accurate predictions about the entities involved compared to models that only consider individual entities in isolation, such as a standard multi-layer perceptron (MLP). Using a GNN also allows adaptation to changes in topology with only minor adjustments. In contrast, an MLP, the standard in deep learning-based ACOPF literature, would require a complete retrain. Topology adaptability is essential in real-world scenarios where line and generator outages or other unexpected changes to topology can occur. Only one study we know of has used GNN for ACOPF, [9], which focused on computing ACOPF solutions directly with the GNN. Although warm-start solutions for iterative solvers using machine learning and deep learning have been explored in [7,10–13] these methods do not use reinforcement learning to factor in the convergence time of the employed iterative solver into the loss function of the learning algorithm and also do not use GNN to take advantage of the graph structure of the power system.

With the proposed framework, we aim to demonstrate a middle-ground solution to ACOPF that combines the speed of a trained neural network with the accuracy of iterative solvers. This framework is evaluated on the feasibility of solutions and the computation time required to compute solutions. We conduct experiments using standard IEEE test environments.

The contributions of our work are as follows:

(1) We create a novel framework for solving ACOPF utilizing GNN and PPO as learning mechanisms.
(2) We introduce a novel deep reinforcement learning reward function for ACOPF that considers the iterative solver's convergence time, which has not yet been explored in ACOPF literature.
(3) We demonstrate state-of-the-art results that do not compromise accuracy or computation time.

By showcasing this novel solution to ACOPF, we aim to provide a new direction for future ACOPF research to compute accurate solutions in real-time.

The structure of this paper is as follows: Section 2 presents the background material necessary for understanding the details of the proposed framework; Section 3 outlines the methodology, including the characteristics of the agent, the GNN architectures, and the structure of the training algorithm; Section 4 provides an overview and discussion of the results obtained from each experiment; and Section 5 provides a brief conclusion of the paper and description of future work.

2. Background

In this section, we review background material that supports the framework presented in this paper. This includes a brief outline and formulation of ACOPF, graph convolutional networks, reinforcement learning, and PPO.

2.1. AC optimal power flow

Given a power system where $\mathcal{N}$ denotes the set of all buses, $\mathcal{G}$ the set of all generators, and $\mathcal{L}$ set of all transmission lines, we formulate the ACOPF as a set of equality and inequality constraints and an objective function to be minimized. Formally,
minimize:

$$\sum_{i\in\mathcal{G}}(C_{2i}P_{gi}^2 + C_{1i}P_{gi}) \tag{1}$$

Subject to:

$$P_{gi}^{min} \le P_{gi} \le P_{gi}^{max}, \qquad \forall i \in \mathcal{G}, \tag{2}$$

$$Q_{gi}^{min} \le Q_{gi} \le Q_{gi}^{max}, \qquad \forall i \in \mathcal{G}, \tag{3}$$

$$V_i^{min} \le V_i \le V_i^{max}, \qquad \forall i \in \mathcal{N}, \tag{4}$$

$$|S_{flow,ij}| \le |S_{flow,ij}^{max}|, \qquad \forall (i,j) \in \mathcal{L}, \tag{5}$$

$$P_i = \sum_{\forall j\in\mathcal{N}} V_iV_j(g_{ij}cos(\theta_j-\theta_i) + b_{ij}sin(\theta_j-\theta_i)), \quad \forall i \in \mathcal{N}, \tag{6}$$

$$Q_i = \sum_{\forall j\in\mathcal{N}} V_iV_j(g_{ij}sin(\theta_j-\theta_i) - b_{ij}cos(\theta_j-\theta_i)), \quad \forall i \in \mathcal{N}, \tag{7}$$

where $C_{1.}$ and $C_{2.}$ are coefficients related to the costs of power generation, P_i and Q_i are the total real and reactive power at bus $i \in \mathcal{N}$, respectively, P_{gi} and Q_{gi} are the total real and reactive generation at generator $i \in \mathcal{G}$, $S_{flow,ij}$ is the apparent power of line $(i,j) \in \mathcal{L}$, V_i and θ_i are the voltage magnitude and the phase angle at bus $i \in \mathcal{N}$, respectively, g_{ij} and b_{ij} are the conductance and susceptance, respectively, of the line $(i,j) \in \mathcal{L}$ that connects buses i and j.

(1) is the objective function to be minimized, relating to power generation costs. (2)–(5) are inequality constraints of the ACOPF related to limits on power generation, voltage limits, and line flow limits. (6) and (7) are equality constraints related to power balance.

2.2. Graph convolutional networks

Given a graph with a set of nodes or vertices V and a set of edges that denote connections between nodes E, GNNs use optimizable operations, such as aggregation [14], pooling [15], attention [16], or convolution [17], that consider entities of a graph as interconnected rather than independent, taking into account their relationships with one another. The identity of a node or edge can be represented by one or more values; for example, in the context of power systems, nodes can denote buses and be represented by values such as real power, reactive power, voltage, etc. In contrast, edges can represent the lines that connect two buses and be represented by values such as resistance, conductance, susceptance, current, etc. Graph-structured data exists in various domains, from street traffic to molecular biology to social networks. Prediction tasks for graph-structured data can fall into three categories: graph-level, node-level, or edge-level. Graph-level tasks involve computing all properties of an entire graph; node-level

tasks are concerned with predicting values that represent the identity of a node; and edge-level tasks can involve predicting where edges in a graph should be or determining the values that represent that edge's identity. The ACOPF problem outlined in this paper will be a node-level task.

Graph convolutional networks, a popular type of GNN, utilize an operation known as graph convolution, which is similar to the convolution operation used in a convolutional neural network for images; the new representation of a node in a graph becomes an amalgamation of itself and its neighbors. We compute a transformation of each node using feature information from all its neighbors and itself, allowing the network to consider nodes as interconnected rather than independent entities. The convolution operation translates well between images and graphs simply because images are graphs, too, where each pixel is nodally connected to adjacent pixels. While the idea of graph convolution is similar to the convolution operation used in a convolutional neural network for images, their mathematical formulations differ; also, graphs require an order-invariant operation, whereas the convolution operation used for images is not order-invariant. In this study, we adopt the graph convolution formulation in [18], shown below:

$$\mathbf{x}_i' = \mathbf{W}_1\mathbf{x}_i + \mathbf{W}_2 \sum_{j\in\mathcal{N}(i)} e_{j,i}\cdot\mathbf{x}_j, \tag{8}$$

where $x_{(\cdot)}$ is the value(s) of a node, $e \in \mathbb{R}^{\|\mathcal{N}\|\times\|\mathcal{N}\|}$ is the adjacency matrix, where $\|(\cdot)\|$ denotes cardinality of the set, and $W_{(\cdot)}$ are learned weight matrices.

2.3. Reinforcement learning

Reinforcement learning is a collection of optimization algorithms where an agent aims to learn the optimal strategy for interacting with its environment based on trial-and-error learning [19]; the environment describes the world with which the agent interacts. Reinforcement learning algorithms can be categorized into model-based or model-free algorithms; in model-based approaches, the agent uses a predictive model of the environment to determine the possible outcomes of its actions. Model-free approaches, which do not use predictive models of the environment to guide the agent's decision-making process, rely more on the expected returns of their actions to understand their environment. Model-free reinforcement learning can be divided into two main categories: value-based and policy-based. Value-based approaches aim to learn or estimate the value or expected return of state–action pairs. In policy-based approaches, the agent devises a policy such that the action performed in every state helps the agent gain maximum future reward.

At each time step t, the agent receives observations from the environment and takes actions based on those observations. The environment then responds with a new state and a reward signal, which the agent uses to update its policy or value function. This loop continues until the agent reaches its goal or the environment ends. Using a neural network, the agent can progressively learn a model of the optimal policy or value function and eventually be capable of taking actions that would earn the largest long-term return.

A Markov decision process can be used to model reinforcement learning problems. This consists of four elements: the state space of the environment s, the action space of the environment a, the reward function r, and the transition probability p. The state space encompasses any information from the agent's environment that can and should be used to make decisions. The actions space includes the agent's decision variables — the set of all possible actions. The reward function defines the reward given to the agent after an action is executed and the results of the action are realized. Rewards can also have zero or negative values, often denoting punishment. Lastly, the transition probability delineates the stochasticity of the environment: the probability of a specific state s_{t+1} being observed when action a_t is taken at a specific state s_t. If the probability is 1, the environment would be deterministic.

2.4. Proximal policy optimization

As a derivative of policy gradient and trust region methods, PPO is a policy-based reinforcement learning algorithm that employs elements from each to reach a balance between ease of implementation, sample complexity, and ease of tuning to compute an update that minimizes the cost function while ensuring the deviation from the previous policy is relatively small [8]. We use PPO because policy-based methods generally work better in environments with infinitely large state and action spaces, as value-based cannot reasonably obtain accurate mappings of optimal state–action pairs without significant training.

Algorithm 1 outlines PPO's structure. In each episode of training, for T time steps (where T is a much smaller value than the length of an episode[2]), the agent will use the same policy; once the number of time steps reaches T, the agent will undergo a policy update. PPO uses an actor–critic structure: the actor learns the optimal policy, whereas the critic estimates the value function, which computes the potential long-term return. The actor and the critic are both modeled by a neural network.

Algorithm 1 PPO

```
for iteration = 1, 2, … do
    Run policy π_θold in environment for T timesteps
    Compute advantage estimates Â_1, …, Â_T
    Optimize surrogate L wrt θ, with K epochs and
    minibatch size M ≤ NT
  θ_old ← θ
end for
```

The objective function to be minimized by the agent is defined as:

$$L_t(\theta) = L_t^{clip}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right]\left(s_t\right), \tag{9}$$

where c_1 and c_2 are user-defined coefficients to determine the weight of each component in the objective function, $L_t^{VF}(\theta)$ is the loss of value function (the critic network), L^{clip} is the clipped surrogate loss, and S is the entropy of the policy. The policy, π_θ, is defined by a multivariate Gaussian distribution $\pi_\theta \sim Norm\left(\boldsymbol{\mu}_\theta(s_t), \boldsymbol{\Sigma}_{\pi\theta}\right)$. The mean of the distribution, $\boldsymbol{\mu}_\theta(s_t)$, is the output produced by the actor network, and $\boldsymbol{\Sigma}_{\pi\theta}$ is the diagonal matrix of the covariance matrix with a user-defined standard deviation; the standard deviation is set relatively high for earlier episodes and decays as the episodes progress, adding small amounts of random noise to the action and aiding in exploration.

The clipped surrogate loss is defined as

$$L^{clip}(\theta) = \min(R_t(\theta)\hat{A}_t^{\pi_{\theta,\ \text{old}}}, \quad \text{clip}\left(R_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t^{\pi_{\theta,\ \text{old}}}), \tag{10}$$

where θ_{old} refers to the policy parameter before the actor is updated, $\hat{A}_t$ is the advantage estimate, which is defined as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{11}$$

where

$$\delta_t = R_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right), \tag{12}$$

and $R_t(\theta)$ is the ratio between the new and old policies, defined as:

$$R_t(\theta) = \pi_\theta\left(a_t \mid s_t\right)/\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right). \tag{13}$$

The clip function ensures that $R_t(\theta)$ is between $1-\varepsilon$ and $1+\varepsilon$, where ε is a user-defined parameter to determine the tightness of the clip boundaries. In (11), variables γ and λ are user-defined variables that control the reward discount; larger values allow the agent to consider long-term rewards, while smaller values enable the agent to consider

[2] An episode refers to a series of agent-environment interactions between the initial state and the terminal state.

shorter-term rewards, and $V(\cdot)$ is the value function modeled by the critic network.

The agent's training generally concludes when the objective function (9) reaches a point of convergence, and the loss is no longer decreasing.

3. Methodology

Firstly, we will define the reinforcement learning problem by outlining the state space, reward function, and action space. The state of the environment s, the agent's input, includes all active and reactive load variables for all buses in the network that belong in $\mathcal{N} = \{1, \dots, N\}$:

$$s = [P_{l1}, \dots, P_{lN}, Q_{l1}, \dots, Q_{lN}], \tag{14}$$

where P_{li} is the real power demand at bus i and Q_{li} is the reactive power demand at bus i. The agent will then execute an action using this state information to inform its decision. The action space is comprised of variables related to power generation (for all generators that belong in $\mathcal{G} = \{1, \dots, G\}$), voltage magnitude, and voltage angles:

$$a = [P_{g1}, \dots, P_{gG}, Q_{g1}, \dots, Q_{gG}, V_i, \dots, V_N, \theta_i, \dots, \theta_N]. \tag{15}$$

Note that the agent's goal is not necessarily to produce a set of actions that result in a feasible and cost-minimizing solution to the ACOPF, so the reward function must reflect the agent's true goal to compute a set of actions that minimizes the amount of time required for the IPS to converge to a solution. Thus, we designed the following reward function:

$$r = \begin{cases} -r_{nc}, & \text{IPS could not converge given } a_t \\ -r_c, & \text{IPS converges given } a_t \ , \end{cases} \tag{16}$$

where r_{nc} is a fixed value denoting the value given to a non-convergence, and r_c is the amount of time, in seconds, it takes for the IPS solver to converge given a. While non-convergence can often result from a truly unsolvable state, they are very likely to result from a poor initial guess as IPS can be very sensitive to the given initial guess. Thus, we must give the agent a punishment if the IPS does not converge given a. This punishment is not arbitrarily selected; it must be determined carefully as it should not overpower the convergence rewards; otherwise, the agent may not learn to minimize IPS time and too heavily prioritize minimizing non-convergence. Also, it cannot be too small since some more difficult states may take up to a few seconds to solve, and the punishment for non-convergence should always be a lesser value than a convergence reward.

3.1. Environmental set-up

The environment is constructed using PyPower, the official Python port of MATPOWER. Experiments are conducted using Python 3.7. computations are conducted on an NVIDIA A100 40 GB GPU.

The proposed framework is trained and evaluated on the IEEE 14-bus, IEEE 118-bus, and IEEE 300-bus systems. As a baseline, we will compare results to PyPower's ACOPF solver, which computes initial guesses as an average of the minimum and maximum values of each variable in the action space. PyPower's IPS is used for all experiments. We also compare two deep learning methods to compute initial guesses: a GNN trained to produce feasible and cost-minimizing ACOPF solutions and a multi-layer perceptron in conjunction with PPO trained to produce solutions that minimize IPS convergence time. These baselines can also act as an ablation study to validate the claims that, for the purpose of computing a warm-start:

(1) A loss function designed to minimize IPS convergence time will be more effective than a loss function designed to produce a feasible and cost-minimizing solution.

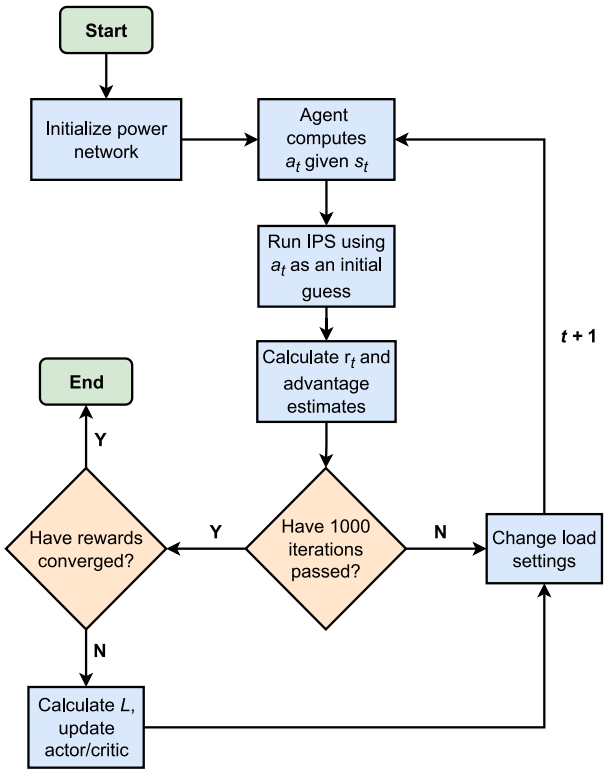

Fig. 1. A flowchart of the training algorithm.

(2) A neural network architecture that can exploit the graph structure of a power system by accounting for relations between nodes will be more effective than a neural network architecture that treats nodes as independent entities.

The baseline GNN will have the same architecture as the GNN in the proposed PPO-GNN framework, but it will instead use a mean-squared error loss function and will be trained on a dataset with 20,000 unique states, where the labels are generated by the PyPower IPS solver. As for the PPO-MLP baseline, the loss function will be the same as that in the PPO-GNN implementation, but the neural network architectures will differ.

The structure of the training algorithm is as follows[3]: first, the load at each load bus is adjusted. This adjustment is done by randomly generating a value between 10% and 80% of the system generation limits for the total load across all load buses and then assigning a portion of that total load to each load bus using a Dirichlet distribution, which is defined as:

$$f(z, \boldsymbol{\alpha}) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{i=1}^{D} z_i^{\alpha_i - 1}, \tag{17}$$

$$\mathrm{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{D} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{D} \alpha_i\right)}, \tag{18}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^D$ is a vector of concentration parameters for each load bus, and D is the number of loads in the power network. The Dirichlet distribution is a distribution over a vector $z \in \mathbb{R}^D$ such that each $z_i > 0$ and:

$$\sum_{i=1}^{D} z_i = 1. \tag{19}$$

[3] github.com/AzadDeihim/ACOPF-PPO-GNN.

Table 1
Layer sizes for each neural network architecture. These values denote the dimensionality of the output of each layer, determined by the shape of the weight matrices between each layer.

GNN layer	14-Bus		118-Bus		300-Bus	
	Actor	Critic	Actor	Critic	Actor	Critic
Layer 1 size	128	128	256	256	512	512
Layer 2 size	256	256	512	512	1024	1024
Layer 3 size	128	128	256	256	512	512
MLP layer	14-Bus		118-Bus		300-Bus	
	Actor	Critic	Actor	Critic	Actor	Critic
Layer 1 size	128	128	256	256	512	512
Layer 2 size	256	256	512	512	1024	1024
Layer 3 size	128	128	256	256	512	512

Table 2
Coefficient settings for the objective function of PPO, learning rate settings for actor and critic networks, and discount factor settings.

	14-bus	118-bus	300-bus
c_1	0.5	0.5	0.5
c_2	0.01	0.01	0.01
γ	0.0	0.0	0.0
λ	0.0	0.0	0.0
r_{nc}	12	12	15
Learning rate	0.0005	0.0005	0.0005

The vector returned by Dirichlet distribution denotes the share of the total real power demand each load bus will be delegated. The reactive power demand at each load bus is calculated via a randomly generated power factor between 0.8 and 1.

Given the new state of the environment, the agent will use these values to compute an initial guess. Then, a timer is started, and the initial guess is given to the PyPower IPS as a starting point. When the IPS converges, the timer is stopped. If it converges, the negative of the elapsed time is given as a reward. If it does not converge, a large negative reward is given. After 1000 iterations of this process, the agent will undergo a policy update and then continue from the first step. Statistics regarding average reward, IPS convergence time, and the number of IPS non-convergences are recorded during each 1000 time step period. The total number of iterations is not predefined; training will end once the agent's reward converges and is no longer improving. A flowchart diagram of the training algorithm is shown in Fig. 1.

Separate GNNs model the actor and critic, and their architecture only varies slightly. While they are both given the same input, they will output different values: the actor will output $a \in \mathbb{R}^{2G+2N}$, the action, whereas the critic will output the value or expected reward of state transition. In both the 14-bus experiment and the 118-bus experiment, the actor and critic will have three graph convolution layers, where each layer performs the node-wise transformation described in (8). The output for all nodes of the final graph convolution is flattened into a vector and then linearly transformed via:

$$a = WX + b, \tag{20}$$

where W is the weight matrix and b is the bias vector. We use a hyperbolic tangent activation function between all layers of the GNNs. The optimization algorithm used in backpropagation is the Adam optimizer.

Details regarding hyperparameter settings for each experiment, including the architecture of the actors and critics, can be found in Tables 1 and 2. The action standard deviation is initially set to 0.5 and decays by 0.01 every 250 time steps until it reaches 0. We chose discount factors of 0.0 because actions in different time steps are completely independent, and state transitions are entirely stochastic; thus, long-term rewards should not matter to the agent.

Table 3
Comparison of the mean and standard deviation of IPS convergence times and convergence rate of IPS between PPO-GNN and the baseline on the 14-bus test case.

	Mean r_c	Standard deviation r_c	Convergence %
PPO-GNN	0.53 s	0.09 s	100.0%
PyPower	0.79 s	0.19 s	96.5%
GNN	0.62 s	0.13 s	97.9%
PPO-MLP	0.61 s	0.09 s	96.2%

4. Results

This section presents results from the experiments outlined in Section 3.1. PPO-GNN and the baseline are evaluated on the same 5000 randomly generated states for all experiments. During testing, the critic network is discarded as it is only required for training; all initial guess computation is conducted by the actor. All models will be evaluated on their ability to produce initial guesses that can minimize IPS convergence time and minimize the number of non-convergences.

4.1. 14-Bus

The IEEE 14-bus test case represents a small power system with 14 buses, five generators, 11 loads, and 20 lines. For the IEEE 14-bus system, we demonstrate that our proposed framework can significantly decrease IPS computation time while minimizing the number of non-convergences. Training details are provided in Fig. 2. Fig. 2 shows that after about 9000 time steps, the agent converges to an average reward of −0.54 and is able to compute initial guesses that allow the IPS to solve the ACOPF in under 0.55 s while only causing the IPS to not converge in less than 0.1% of states. We compare the results to the baselines in Table 3. This shows that the proposed framework outperforms the baselines significantly in terms of IPS convergence time and the number of non-convergences. By using the proposed PPO-GNN-based framework to compute initial guesses for IPS, we observe a 33% speed-up over the PyPower ACOPF solver, a 14% speed-up over the GNN, and a 13% speed-up over PPO-MLP. The initial guesses supplied by PPO-GNN resulted in a 100.0% convergence rate on the test set. On the other hand, initial guesses provided by the baselines incurred significantly lower convergence rates. Results in Fig. 2 and Table 3 indicate that the quality of the initial guess plays an important role in the IPS's ability to converge. In the early training time steps, we observe convergence rates below 75%. We also observe that over the course of training, the average IPS convergence time decreases from 0.85 to 0.53 s, demonstrating that even in some of the earlier time steps where PPO-GNN was undertrained, the initial guesses still resulted in lower IPS convergence times than the baselines. Additionally, it only takes the GNN roughly eight milliseconds to compute an initial guess. The total training time for PPO-GNN in this experiment was roughly five hours, the majority of which was IPS-solving time.

4.2. 118-Bus

The IEEE 118-bus test case represents a power system with 118 buses, 19 generators, 35 synchronous condensers, 177 lines, nine transformers, and 91 loads. Similar to the 14-bus system, our framework delivers improved performance over the baselines. We compare the results of PPO-GNN with the baselines in Table 4, demonstrating a 30% improvement over the PyPower solver, a 12% improvement over the GNN, and a 14% improvement over the PPO-MLP in terms of IPS computation time. Fig. 3 shows a detailed visualization of training statistics. After about 12,000 time steps, the agent converged to an average reward of −1.19, an average IPS convergence time of 1.19 s, and a 100.0% convergence rate. Similar to the 14-bus experiment, PPO-GNN is the only model capable of achieving a 100.0% convergence rate on the test set. Also, during training, we observed much lower

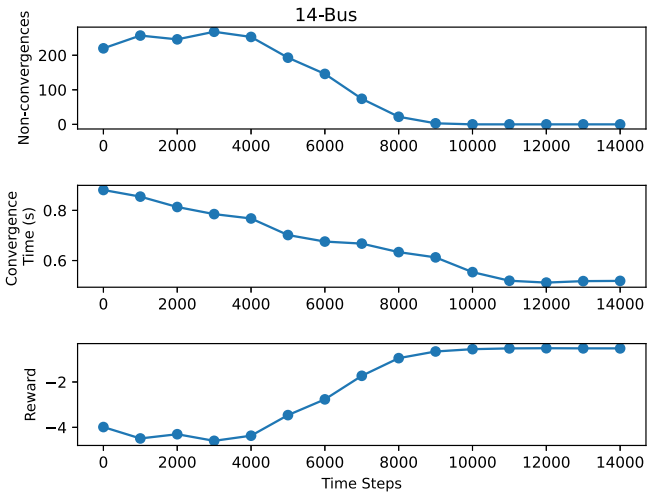

Fig. 2. PPO-GNN training statistics for 14-bus test case. The three scatter plots represent the number of non-convergences, average IPS convergence time, and average reward on the y-axis and the number of time steps on the x-axis.

Table 4
Comparison of the mean and standard deviation of IPS convergence times and convergence rate of IPS between PPO GNN and the baseline on the 118-bus test case.

	Mean r_c	Standard deviation r_c	Convergence %
PPO-GNN	1.19 s	0.19 s	100.0%
PyPower	1.70 s	0.39 s	99.4%
GNN	1.35 s	0.23 s	99.6%
PPO-MLP	1.39 s	0.28 s	97.9%

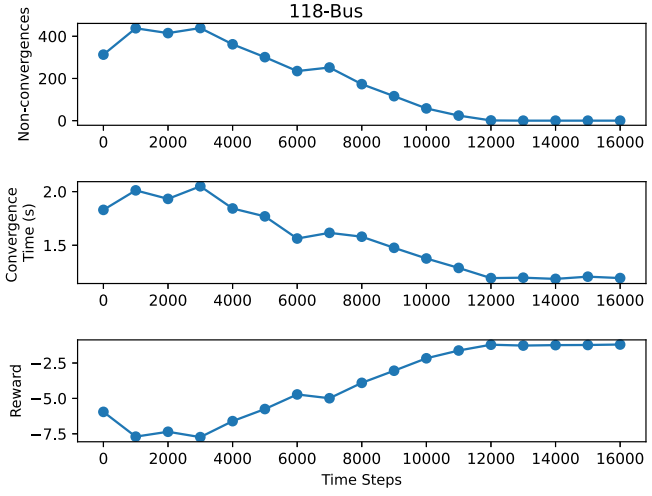

Fig. 3. PPO-GNN training statistics for the 118-bus test case. The three scatter plots represent the number of non-convergences, average IPS convergence time, and average reward on the y-axis and the number of time steps on the x-axis.

convergence rates in the early stages of training, often below 60%, as well as IPS convergence times of over 2 s, showing that in this experiment, the outcome of the IPS is heavily reliant on the quality of an initial guess. Since the GNN was larger in this test case, it took the GNN about 18 ms on average to compute an initial guess. The total training time for PPO-GNN in this experiment was roughly 12 h, the majority of which is IPS solving time.

4.3. 300-Bus

The IEEE 300-bus system contains 69 generators, 60 load tap changers, 304 transmission lines, and 195 loads. This is the largest and most complex of the power systems used in this study. As shown in Table 5, PPO-GNN outperforms all baselines but only with a marginal improvement over the PyPower ACOPF solver, which in previous experiments performed significantly worse than other models. This most likely indicates that other models performed poorly on this test case, rather than the PyPower ACOPF solver performing well. Fig. 4 displays the training statistics for PPO-GNN on the 300-bus test case. In the early stages of training, we can see that the initial guesses provided resulted in non-convergence nearly 100% of the time but slowly decreased to nearly 0% after roughly 20,000 time steps of training. Similarly, convergence time decreased by a factor of over three from nearly 11 s to under three seconds from the start of training to the end of training. This indicates that the 300-bus test case is significantly more sensitive to the quality of an initial guess compared to smaller test cases, and it is likely that this trend will continue if experiments were to be conducted on larger, more complex systems. Training time for PPO-GNN on the 300-bus test case exceeded training time on other test cases by a substantial margin, requiring nearly 30 h to fully train. Nonetheless, when fully trained, initial guess computation only takes about 30 ms.

Table 5
Comparison of the mean and standard deviation of IPS convergence times and convergence rate of IPS between PPO GNN and the baseline on the 300-bus test case.

	Mean r_c	Standard deviation r_c	Convergence %
PPO-GNN	2.79 s	0.71 s	99.5%
PyPower	2.98 s	0.84 s	99.3%
GNN	3.35 s	0.91 s	98.1%
PPO-MLP	3.43 s	0.87 s	95.9%

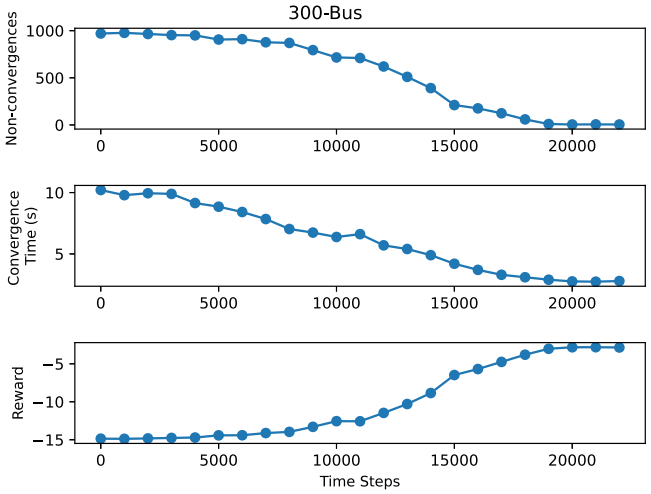

Fig. 4. PPO-GNN training statistics for the 300-bus test case. The three scatter plots represent the number of non-convergences, average IPS convergence time, and average reward on the y-axis and the number of time steps on the x-axis.

4.4. Supplementary experimentation

We conduct additional experiments by sampling a wider range of initial guesses to observe how an initial guess may affect the quality of the resulting IPS convergence point in terms of cost, the ability to converge, and convergence time. This experiment also aims to understand if the reward function presented in (16) may cause the IPS to converge quickly but to a suboptimal solution. We sample a uniform distribution within the system's limits of 2000 unique initial guesses; we then record the resulting solution's costs and the IPS's convergence time, given these initial guesses, if it could converge. This experiment was conducted using the same random state on each test case. Results of this experiment are presented in Table 6. In every case, the IPS

Table 6
Convergence rate, convergence time, and resulting cost of IPS given a random uniform distribution of initial guesses.

	Convergence %	Min. Cost $	PPO-GNN cost $	Mean convergence time	PPO-GNN convergence time
14-Bus	76.7%	2.03E+07	2.03E+07	0.82 s	0.49 s
118-Bus	93.8%	3.63E+08	3.63E+08	1.50 s	1.15 s
300-Bus	34.2%	2.38E+09	2.38E+09	5.10 s	3.02 s

converged to the same point regardless of the initial guess, if it could converge. Still, convergence rates in this experiment were far lower than in previous experiments as initial guesses were randomly selected, indicating that IPS may be more sensitive to the quality of an initial guess than what was suggested in our previous experiments; the same can be said for IPS computation time. These results also suggest that the PPO-GNN can provide an initial guess that produces a fast solution without compromising the quality of the resulting ACOPF solution.

5. Conclusions

In this work, we proposed a novel framework for attaining quick, optimal solutions to ACOPF using PPO and GNN to compute initial guesses for an IPS. These initial guesses were assessed by whether or not the IPS could converge, given that initial guess and how quickly it could converge to the optimal solution. This resulted in a trained GNN that produces IPS initial guesses that minimize the number of non-convergences and significantly decrease IPS convergence time compared to three baselines: PyPower's ACOPF solver, initial guesses computed by a GNN that is trained to calculate ACOPF solutions directly, and an MLP trained with PPO using the same reward function as our proposed framework. This was demonstrated on three different IEEE test environments: the 14-bus test case, the 118-bus test case, and the 300-bus test case. In the future, we look to explore the use of this framework on much larger power system test cases to better understand the scalability of this solution, as well as instances of topology changes, such as line outages. Additionally, this work did not include a comprehensive assessment of alternative GNN architectures or different reinforcement learning algorithms, nor did we explore a wide variety of hyperparameter settings. With that, the solution presented in this paper can be improved significantly, and we hope that this can provide a new direction for ACOPF research.

CRediT authorship contribution statement

Azad Deihim: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing. **Dimitra Apostolopoulou:** Supervision, Writing – review & editing. **Eduardo Alonso:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

There is no data, but there is code. A github link is included.

References

[1] E. Mohagheghi, M. Alramlawi, A. Gabash, P. Li, A survey of real-time optimal power flow, Energies (2018).
[2] K. Baker, Solutions of DC OPF are never AC feasible, in: Proceedings of the Twelfth ACM International Conference on Future Energy Systems, Association for Computing Machinery, 2021, pp. 264–268.
[3] Y. Zhou, B. Zhang, C. Xu, T. Lan, R. Diao, D. Shi, Z. Wang, W.-J. Lee, A data-driven method for fast AC optimal power flow solutions via deep reinforcement learning, J. Mod. Power Syst. Clean Energy (2020) 1128–1139.
[4] L. Zhang, B. Zhang, Learning to solve the AC optimal power flow via a Lagrangian approach, in: North American Power Symposium, 2022, pp. 1–6.
[5] Z. Wang, J.-H. Menke, F. Schäfer, M. Braun, A. Scheidler, Approximating multi-purpose AC optimal power flow with reinforcement trained Artificial Neural Network, 2022.
[6] R. Nellikkath, S. Chatzivasileiadis, Physics-informed neural networks for AC optimal power flow, Electr. Power Syst. Res. (2022).
[7] Y. Cao, H. Zhao, G. Liang, J. Zhao, H. Liao, C. Yang, Fast and explainable warm-start point learning for AC Optimal Power Flow using decision tree, Int. J. Electr. Power Energy Syst. (2023) 109369.
[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, CoRR.
[9] S. Liu, C. Wu, H. Zhu, Topology-aware graph neural networks for learning feasible and adaptive AC-OPF solutions, IEEE Trans. Power Syst. (2022) 1–11.
[10] K. Baker, Learning warm-start points for Ac optimal power flow, in: IEEE 29th International Workshop on Machine Learning for Signal Processing, 2019, pp. 1–6.
[11] X. Pan, M. Chen, T. Zhao, S.H. Low, DeepOPF: A feasibility-optimized deep neural network approach for AC optimal power flow problems, IEEE Syst. J. (2023) 673–683.
[12] R. Canyasse, G. Dalal, S. Mannor, Supervised learning for optimal power flow as a real-time proxy, in: IEEE Power & Energy Society Innovative Smart Grid Technologies Conference, 2017, pp. 1–5.
[13] P.L. Donti, D. Rolnick, J.Z. Kolter, DC3: A learning method for optimization with hard constraints, 2021, CoRR.
[14] W.L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1025–1035.
[15] Z. Ma, J. Xuan, Y.G. Wang, M. Li, P. Liò, Path integral based convolution and pooling for graph neural networks, in: Advances in Neural Information Processing Systems, 2020, pp. 16421–16433.
[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, 2018, CoRR.
[17] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, CoRR.
[18] C. Morris, M. Ritzert, M. Fey, W.L. Hamilton, J.E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, 2019.
[19] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, second ed., The MIT Press, 2018.