PowerFlowNet

Power flow approximation using message passing Graph Neural Networks

Lin, Nan; Orfanoudakis, Stavros; Cardenas, Nathan Ordonez ; Giraldo, Juan S.; Vergara, Pedro P.

DOI
10.1016/j.ijepes.2024.110112

Publication date
2024

Document Version
Final published version

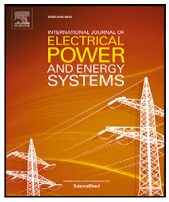
Published in
International Journal of Electrical Power and Energy Systems

Electrical Power and Energy Systems 160 (2024) 110112

Contents lists available at ScienceDirect

International Journal of Electrical Power and Energy Systems

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



PowerFlowNet: Power flow approximation using message passing Graph Neural Networks

Nan Lin [a,*], Stavros Orfanoudakis [a,*], Nathan Ordonez Cardenas [b], Juan S. Giraldo [c], Pedro P. Vergara [a]

[a] Delft University of Technology, Intelligent Electrical Power Grids, Mekelweg 5, Delft, 2628 CD, The Netherlands
[b] Delft University of Technology, Software Technology, Van Mourik Broekmanweg 6, Delft, 2628 XE, The Netherlands
[c] Energy Transition Studies Group, Netherlands Organization for Applied Scientific Research (TNO), Radarweg 60, Amsterdam, 1043 NT, The Netherlands

ARTICLE INFO

Keywords:
Power flow
Deep learning
Machine Learning
Data driven

ABSTRACT

Accurate and efficient power flow (PF) analysis is crucial in modern electrical networks' operation and planning. Therefore, there is a need for scalable algorithms that can provide accurate and fast solutions for both small and large scale power networks. As the power network can be interpreted as a graph, Graph Neural Networks (GNNs) have emerged as a promising approach for improving the accuracy and speed of PF approximations by exploiting information sharing via the underlying graph structure. In this study, we introduce PowerFlowNet, a novel GNN architecture for PF approximation that showcases similar performance with the traditional Newton–Raphson method but achieves it 4 times faster in the IEEE 14-bus system and 48 times faster in the realistic case of the French high voltage network (6470rte). Meanwhile, it significantly outperforms other traditional approximation methods, such as the DC power flow, in terms of performance and execution time; therefore, making PowerFlowNet a highly promising solution for real-world PF analysis. Furthermore, we verify the efficacy of our approach by conducting an in-depth experimental evaluation, thoroughly examining the performance, scalability, interpretability, and architectural dependability of PowerFlowNet. The evaluation provides insights into the behavior and potential applications of GNNs in power system analysis.

1. Introduction

The complexity of electrical power systems is continuously rising, largely attributed to the substantial integration of decentralized renewable energy resources. Within this context, power flow (PF) stands as a fundamental challenge in ensuring the stability of power systems, playing a pivotal role in both the operational management and long-term planning of electrical networks. At its core, PF is a mathematical problem that revolves around determining the voltages at various buses, a task accomplished by solving a set of nonlinear equations, which are inherently linked to the network configuration, load distribution, and generation characteristics [1]. Traditional methods, such as Newton–Raphson [2], the Gauss–Seidel [3], and the fast-decoupled methods, have excellent accuracy and convergence properties but scale slowly with larger power systems, particularly in long-term planning of national grids with thousands of buses [4]. In contrast, the DC power flow (DCPF) technique [5] simplifies this problem into a linear one by making assumptions about the voltage magnitude, but at the cost of accuracy. Consequently, there is a need for innovative algorithms capable of efficiently solving the PF for significantly larger networks. These novel algorithms should aim to balance the need for speed without compromising accuracy, ultimately facilitating the streamlined operation and planning of large real-world electrical grids.

Modernizing the grids with the addition of accurate smart metering and data acquisition systems has enabled the development of Machine Learning (ML) methods for accurate and efficient power system analysis [6]. ML PF methods use historical operation data gathered from the metering infrastructure and try to approximate the power system state based on them. In [7], the authors effectively transformed the nonlinear PF relationship into a linear mapping within a higher-dimensional state space, resulting in a substantial improvement in the precision of the calculation process. Similarly, Chen et al. [8] mitigate the errors of model-based PF linearization approaches by approximating the nonlinear PF equations in a data-driven manner. In [9], the data-driven PF method comprises two distinct stages: offline learning and online computing. During the offline learning stage, a learning model is developed utilizing the proposed exact linear regression equations,

* Corresponding authors.
E-mail addresses: n.lin@tudelft.nl (N. Lin), s.orfanoudakis@tudelft.nl (S. Orfanoudakis).

https://doi.org/10.1016/j.ijepes.2024.110112
Received 26 February 2024; Received in revised form 14 May 2024; Accepted 24 June 2024
Available online 3 July 2024

Table 1
ML-based methods for state estimation and PF.

Study	Problem	Neural network architecture	Max. Experiment scale (Buses)
[12]	Grid topology and PF	MLP + Bilinear NN	2383
[13]	Probabilistic PF	MLP	661
[14]	Grid topology and PF	Deep shallow NN	123
[15]	State estimation	Gauss–Newton unrolled NN + GNN	118
[16]	State estimation	GCN + GRN	123
[17]	State estimation	Hyper-Heterogeneous multi graph GNN	179
[18]	PF	Iterative GNN-based	118
Ours	PF	Mask encoder + GCN + Message passing	6470

which is subsequently solved by applying the ridge regression method to mitigate the impact of data collinearity. Subsequently, the need for nonlinear iterative calculations is obviated in the online computing stage, thereby streamlining the computation process. Furthermore, Liu et al. [10] reformulated PF into a regression model, leveraging the structural attributes of AC power flow equations, including Jacobian matrix-guided constraints, to substantially reduce the search space. Overall, all these simple ML methods have outperformed the DCPF method [11]; however, the scalability issues still do not allow for the application of these methods in large power systems.

Numerous deep learning techniques have been developed, promising to find more scalable solutions. Many studies emphasize the use of physics-informed methodologies that leverage the inherent characteristics of the problem, as demonstrated in Table 1. For example, Hu et al. [12] incorporate an auxiliary task for PF model reconstruction, wherein the neural network (NN) based PF solver is effectively regularized by encoding varying levels of Kirchhoff's laws and system topology into the reconstructed PF model, consequently ensuring adherence to physical laws and constraints. In [13], the training process of the NN is enhanced through the physical PF equations, such as incorporating branch flows as a penalty term in the NN's objective function and simplifying the backpropagation update gradients based on the transmission grid's physical characteristics. Similarly, Li et al. [14] use NNs to estimate the distribution system state equations by creating virtual nodes to represent buses without any smart metering. An alternative approach is to enhance a classical solver (Gauss–Newton) by training an NN to serve as a regularizer [15]. However, it is important to note that classic deep learning approaches do not exploit information sharing via the underlying graph structure of power system, leading to limited generalization capabilities and efficiency in real-world large-size network scenarios.

GNNs [19] have garnered significant attention in recent years due to their capacity to leverage graph-structured data by aggregating information from neighboring nodes. In practice, GNNs are efficient because the same parameters can be used for every node. Consequently, researchers have explored diverse GNN architectures to tackle the PF problem because the topology of a power system can be naturally interpreted as a graph. In particular, GNNs have been successful in various tasks related to power systems [20], such as fault detection, time-series prediction, power flow calculation, and data generation. Wu et al. [16] used the power flow equations to derive a novel representation of the power system topology, which can be utilized in GNNs to improve performance in power grid state forecasting and voltage control through reinforcement learning. In [17], a weakly supervised learning approach is employed based on the PF equations that do not require labeled data, however, at the cost of accuracy compared to fully supervised methods. Meanwhile, [18] improves upon an iterative-based GNN approach by employing a purely physics-constrained model loss.[1] Despite the proliferation of various studies in this domain, little attention has been directed toward investigating the effectiveness of proposed methodologies for large power networks. Consequently, a comprehensive evaluation and comparison of traditional and GNN-based approaches concerning accuracy, computational complexity, scalability, and interpretability are imperative to discern the most promising techniques for PF analysis.

To address the scalability and other aforementioned real-world application concerns, we propose PowerFlowNet. PowerFlowNet is a novel GNN architecture specifically designed to leverage electrical power networks' structural characteristics and interconnectedness, enabling efficient approximation of the PF. In practice, PowerFlowNet transforms the PF into a GNN node-regression problem by representing each bus as a node and each transmission line as an edge while maintaining the network's connectivity. The key advantage of PowerFlowNet lies in its significantly lower execution time, regardless of the network size; thus, enabling it to be applied for realistic power grid planning and operation processes. Specifically, our thorough experimental evaluation demonstrates PowerFlowNet's accuracy and execution speed compared to traditional methods, such as DCPF, while its performance is similar to the most trusted Newton–Raphson method but with significantly lower execution time. During planning and operation with distributed energy resources, thousands of PF calculations are needed because of the number of possible scenarios. In such cases, accuracy and low execution time are indispensable. Additionally, an in-depth analysis of the proposed approach's scalability, architectural dependability, and interpretability is conducted, aiming to provide valuable insights into the broader applicability of GNN algorithms in the power system domain.

In detail, the key contribution of this paper is summarized as follows:

- We introduce a novel GNN architecture for PF approximation with significantly lower executing time (during operation) and comparable results compared to the Newton–Raphson method. Its distinctiveness, compared to existing PF GNN approaches, lies in its adept utilization of the capabilities from message passing GNNs [21] and high-order GCN [22], in a unique arrangement called PowerFlowConv, for handling a trainable masked embedding of the network graph. This innovative approach renders the proposed GNN architecture scalable, presenting an effective solution for the PF problem for larger-size networks.
- PowerFlowNet demonstrates the ability to deliver accurate PF approximations for individual buses, using data only from neighboring buses up to four steps away, even in scenarios where not all buses within the system are observed.

The subsequent sections are structured as follows: Section 2 provides a brief review of GNNs, while in Section 3, a comprehensive exposition of PowerFlowNet and of our interpretation of PF as a GNN problem is presented. Lastly, an extensive evaluation of the proposed algorithm followed by an insightful discussion is presented in Section 4.

2. GNNs and classic PF algorithms

This section provides an overview of the fundamental background of GNNs and the methodologies they employ. Furthermore, it delves into the traditional techniques used in PF analysis, including the DCPF method.

[1] In this paper, we distinguish the model loss for training a neural network and the line loss of the power system.

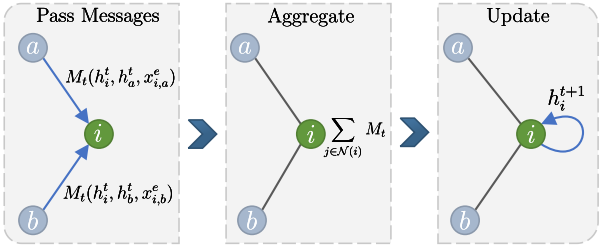


Fig. 1. A message passing step of node i consisting of message creation, aggregation, and update of the hidden state.

2.1. Graph neural networks

GNNs are a class of ML models designed to operate on graph-structured data, which can represent complex relationships and interactions between entities [23]. GNNs have gained significant attention due to their ability to capture and leverage the structural information inherent in graphs. In detail, a graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ can be defined as a set of $\mathcal{N}$ nodes and $\mathcal{E}$ edges with node features[2] $\boldsymbol{x}_i \in \mathbb{R}^F$ and edge features $\boldsymbol{x}_{i,j}^e \in \mathbb{R}^{F_e}$, where $i \in \mathcal{N}$ and the pair $(i, j) \in \mathcal{E}$. Additionally, GNNs can use regular, fully connected layers to help them understand and work with the structure of these graphs.

2.1.1. Message passing NNs

At the heart of GNNs lies the concept of message passing [21]. Message passing allows information to propagate through the graph by iteratively aggregating and updating node features based on their neighborhood relationships. In each message passing step, each node receives information from its neighbors, performs local computations, and then transmits updated information to its neighbors (Fig. 1). Specifically, the message passing phase can run iteratively for T steps. The nodes features in the intermediate layers are often called hidden states. For every node i, its hidden state $\boldsymbol{h}_i^t$ is updated based on the message function $M_t(\cdot)$ and the node update function $U_t(\cdot)$. Therefore, for node i with neighbors $\mathcal{N}(i)$, each iterative message update $\boldsymbol{v}_i^{t+1}$ is

$$\boldsymbol{v}_i^{t+1} = \sum_{j\in\mathcal{N}(i)} M_t(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, \boldsymbol{x}_{i,j}^e), \tag{1}$$

$$\boldsymbol{h}_i^{t+1} = U_t(\boldsymbol{h}_i^t, \boldsymbol{v}_i^{t+1}). \tag{2}$$

As shown in (1), the edge features $\boldsymbol{x}_{i,j}^e$ can also be included in the message creation step. After the message passing phase ends, the final feature vector for the whole graph is denoted as the output of a function $R(\cdot)$

$$\hat{\boldsymbol{y}}(\mathcal{G}) = R(\{\boldsymbol{h}_i^{\mathsf{T}} | i \in \mathcal{G}\}). \tag{3}$$

Notice that the functions $M_t(\cdot), U_t(\cdot)$, and $R(\cdot)$ are designed by the user and should be fully differentiable so that they can be trained and used by graph ML tasks. Overall, message passing enables all nodes to gather and integrate information from their local context, incorporating both the features of neighboring nodes and the graph structure.

2.1.2. Graph convolutional networks

One other popular type of GNN is the GCNs [19], which introduces convolution operations on graphs. Similar to convolution neural networks (CNNs) for regular grid-like data, GCNs apply filters to node features and aggregate information from neighboring nodes. By leveraging the local connectivity of the graph, GCNs can capture both node-level patterns and the information of the K-hop neighbors. In particular, a graph convolutional layer of order K is defined as

$$\hat{\boldsymbol{y}}(S, X) = \sigma\left(\sum_{k=0}^{K-1} S^k X W\right). \tag{4}$$

Here, a GCN layer is a function of the graph shift operator $S \in \mathbb{R}^{N\times N}$, node features $X \in \mathbb{R}^{N\times F}$, and trainable weight matrix $W \in \mathbb{R}^{F\times F}$. Matrix S can be the adjacency matrix A, or a more complex representation such as the Laplacian transformation L. Moreover, σ represents a non-linear function, such as rectified linear units (ReLU), which is employed to enable the approximation of every target non-linear function. The hyper-parameter K holds significance as it dictates the K-hop[3] neighboring nodes are taken into account during each convolution. Consequently, a larger value of K entails the inclusion of more nodes.

2.1.3. Topology adaptive GCN

Many variants of the GCN layer have been proposed with one of them being the topology adaptive graph convolution operator (TAGConv) [22]. TAGConv addresses a limitation of traditional GCN by adaptively learning the importance of different neighboring nodes during message passing. TAGConv assigns weights to the neighbors based on their relevance to the target node. Specifically, a TAGConv layer of order K is defined as

$$\hat{\boldsymbol{y}}(S, X) = \sigma\left(\sum_{k=0}^{K-1} S^k X W_k\right). \tag{5}$$

Here, the trainable weight matrix $W_k \in \mathbb{R}^{F\times F}$ is different for every k. This adaptivity enables TAGConv to effectively capture the local structure and important dependencies in the graph.

2.2. Classic power flow analysis methods

The PF problem is mathematically formulated as solving a set of non-linear equations. A power system has a set of buses (nodes) $\mathcal{N}$ and a set of transmission lines (undirected edges) $\mathcal{E} \subseteq \mathcal{N}\times\mathcal{N}$. For each node $i \in \mathcal{N}$, the PF equations are derived from Kirchhoff's current law and are expressed in complex form as

$$P_i^{\mathrm{L}} - P^{\mathrm{G}} + \mathrm{j}(Q_i^{\mathrm{L}} - Q^{\mathrm{G}}) = \sum_{j\in\mathcal{N}(i)} \dot{V}_i \left(\frac{\dot{V}_j - \dot{V}_i}{z_{i,j}}\right)^* \tag{6}$$

where j is the imaginary unit, italic i, j refer to bus i and j, P_i and Q_i are the active and reactive power drawn from bus i to the ground, $\dot{V}_i = V_i^{\mathrm{m}}\angle\theta_i$ and $\dot{V}_j = V_j^{\mathrm{m}}\angle\theta_j$ are the voltage phasors at bus i and j, $z_{i,j}$ is the impedance of line $(i, j) \in \mathcal{E}$. At each node i, two of P_i, Q_i, V_i^{m}, θ_i are unknown. For a system with $|\mathcal{N}| = N$ nodes, there are $2N$ real-valued equations and $2N$ unknown real-valued variables.

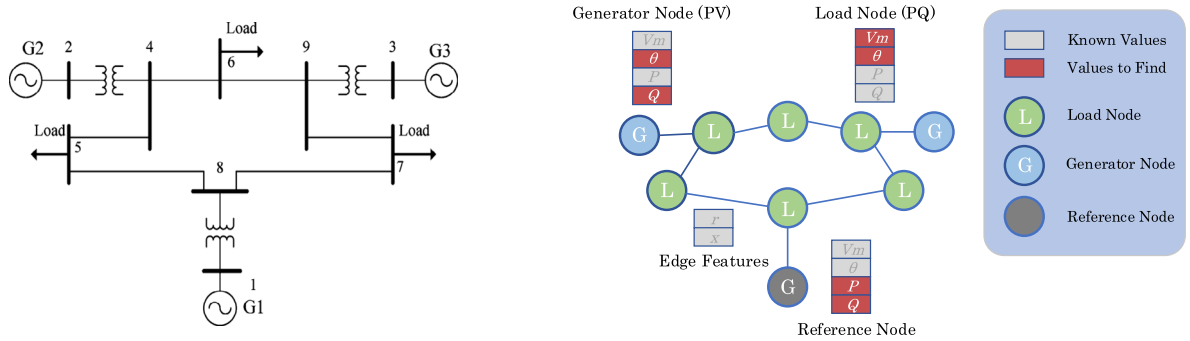
2.2.1. DC power flow approximation

DCPF is an approximation of the ACPF. In DCPF, the voltage is assumed constant at all buses; therefore, the voltage and reactive power differences are 0, so the ΔV^m and ΔQ terms are neglected [5]. Subsequently, the PF equations can be further simplified to a linear problem that does not require an iterative solution. In detail, the relation between the active power (P_i) and the voltage angle (θ_i) of each bus is

$$P_i = \Re\left[\sum_{j\in\mathcal{N}(i)} \dot{V}_i \left(\frac{\dot{V}_j - \dot{V}_i}{z_{i,j}}\right)^*\right] \approx \sum_{j\in\mathcal{N}(i)} \frac{\theta_j - \theta_i}{\Im[z_{i,j}]}. \tag{7}$$

[2] For disambiguation, we always represent the node and edge features with letter x and the line impedance with letter z.

[3] Nodes with distance K from the original node.



(a) Standard IEEE 9-case. (b) GNN equivalent of the IEEE 9-case.

Fig. 2. Interpreting the PF problem of the IEEE 9-case into a GNN node regression problem. In detail, Fig. 2(a) shows the standard IEEE 9-case, and Fig. 2(b) illustrates how each bus is transformed into a load, generator, or reference node while keeping the same line connectivity. Each node has different known and unknown features $\boldsymbol{x} = (V^m, \theta, P, Q)$ depending on its type. The ultimate goal of this GNN problem is to approximate all the features for all nodes, thus solving the PF problem.

3. PowerFlowNet framework

This section presents a thorough explanation of PowerFlowNet, accompanied by our interpretation of PF formulation in the domain of GNN algorithms.

3.1. Interpreting power flow as a GNN problem

The primary objective of a PF study is to determine the voltage magnitudes and angles for a given load, generation, and network state, facilitating subsequent calculations of line flows and power losses. This problem lends itself seamlessly to a GNN approach, as the electrical network topology can be represented as an undirected graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, where the set of nodes $\mathcal{N}$ corresponds to all the buses and the set of edges $\mathcal{E}$ represents power lines. The node features, denoted as $X \in \mathbb{R}^{N\times F}$, encompass various attributes for each bus, including voltage magnitude (V_i^{m}), voltage angle (θ_i), active power (P_i), and reactive power (Q_i). Similarly, the edge features, denoted as $X^e \in \mathbb{R}^{E\times F_e}$, encompass the resistance ($\mathfrak{R}[z_{i,j}]$) and reactance ($\mathfrak{I}[z_{i,j}]$) attributes for each line $(i,j) \in \mathcal{E}$, where $\mathfrak{R}[\cdot]$ and $\mathfrak{I}[\cdot]$ are the real and imaginary part of a complex variable, respectively.

Illustrated in Fig. 2, the transformation of a transmission network into a graph representation involves mapping the buses to nodes while preserving the interconnectedness of the lines. The nodes are categorized into three types: (a) load (PQ) nodes where the P_i and Q_i are known, (b) generator (PV) nodes where V_i^{m} and P_i are known, and (c) a single reference node where both V_i^{m} and θ_i are known. The edge features X^e are assumed to be given. The objective of this task is to predict the missing node features for every node $i \in \mathcal{N}$, e.g., for a PQ node, we need to predict the V^m and θ_m, thereby rendering this problem as a node regression task.

3.2. PowerFlowNet architecture

PowerFlowNet is a GNN approach that reconstructs every node's full feature vector $\hat{\boldsymbol{x}}_i = (V_i^{\mathrm{m}}, \theta_i, P_i, Q_i)$ given partial information of the problem, such as the adjacency matrix of the graph A, the known node features $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^{\mathsf{T}}$, where the unknown features are filled with 0, and the edge features $X^e = [\boldsymbol{x}^e{}_{i,j}]^{\mathsf{T}}$ for each line $(i,j) \in \mathcal{E}$. Our proposed model consists of a mask encoder and a stack of our novel Power Flow Convolutional layers (PowerFlowConv). Initially, the mask encoding layers shift the input features to distinguish known and unknown features using a feature mask. Then, as illustrated in Fig. 3, the encoded graph features are fed to the stack of Power Flow Convolutional operations consisting of a unique arrangement of message passing and TAGConv layers. This way, information from every node and edge is aggregated so that complete feature matrix $\hat{X} = [\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_N]^{\mathsf{T}}$ is predicted. A detailed explanation of PowerFlowNet's components and procedures is presented next.

3.2.1. Mask encoder

In the PF problem, each node has different known and unknown features. The goal is to predict the unknown features while keeping the known ones unchanged. This means that our NN should know which features must be predicted. Consequently, for every input node with feature vector $\boldsymbol{x}_i$ we create a binary mask $\boldsymbol{m}_i \in \mathbb{R}^F$ where 0 represents the known and 1 the unknown features. For example, the mask of a load (PQ) node with a feature vector $\boldsymbol{x}_i = (V_i^m, \theta_i, P_i, Q_i)$ and unknown values V^m and θ, will be $\boldsymbol{m}_i = (1,1,0,0)$. Additionally, we propose using a mask encoder that learns to represent different types of nodes (See Mask Encoder block in Fig. 3). In practice, it consists of two fully connected layers that map a binary mask to a continuous-valued vector. Notably, a fixed (not learned) mask embedding can also be used, but our mask encoder can learn a more flexible mask representation that improves the final performance. The exact mathematical operation for every $m_i \in M$, where $M \in \mathbb{R}^{N\times F}$, is

$$\hat{\boldsymbol{m}}_i = W_1\sigma(W_0\boldsymbol{m}_i + b_0) + b_1, \ \forall i \in \mathcal{N}, \tag{8}$$

which is a function $\boldsymbol{m}_i \in \{0,1\}^F \to \hat{\boldsymbol{m}}_i \in \mathbb{R}^F$, and the weight matrices W_0, W_1 and the biases b_0, b_1 are the trainable parameters. Finally, to produce the encoded graph features X^l, we shift the input node features X with the learned representation $\hat{\boldsymbol{m}}_i$ as

$$\boldsymbol{x}_i^0 = \boldsymbol{x}_i + \hat{\boldsymbol{m}}_i, \ \forall i \in \mathcal{N}. \tag{9}$$

3.2.2. Power flow convolutional layer

The second part of our proposed architecture consists of L connected PowerFlowConv layers that sequentially process the encoded graph features and predict the final feature matrix $\hat{X}^L$, where $\hat{X}^L = [\hat{\boldsymbol{x}}_1^L, \ldots, \hat{\boldsymbol{x}}_N^L]^{\mathsf{T}}$ and $\hat{\boldsymbol{x}}_i^L = (V_i^{\mathrm{m}}, \theta_i, P_i, Q_i)$, $\forall i \in \mathcal{N}$. Each PowerFlowConv layer consists of an initial one-hop message passing step and a K-hop TAGConv (See 2.1.3), which learns to automatically extract information from the edge features $\boldsymbol{x}_{ij}^e$, i.e., the line resistance and reactance, and incorporates it in the neighboring node features $\boldsymbol{x}_i^l$. Message passing exploits information from the edge features, and TAGConv aggregates node features in a large neighborhood. However, in the PF formulation, both line characteristics and node states play an important role. Therefore, by a combination of both techniques the proposed architecture can provide high-quality PF approximations.

More specifically, we calculate the message passed to node $\forall i \in \mathcal{N}$ as

$$\hat{\boldsymbol{x}}_i^l = W_{MP_1}^l \cdot \sigma\left(W_{MP_0}^l \sum_{j\in\mathcal{N}(i)} \langle \boldsymbol{x}_i^l, \boldsymbol{x}_j^l, \boldsymbol{x}_{i,j}^e \rangle + b_{MP_0}^l\right) + b_{MP_1}^l, \tag{10}$$

where $\langle x_i^l, x_j^l, x_{i,j}^e \rangle$ is a concatenated vector of the described vectors fed to a two-layer Multilayer Perceptron (MLP) with an in-between ReLU activation function. Here again, the weight matrices and bias vectors

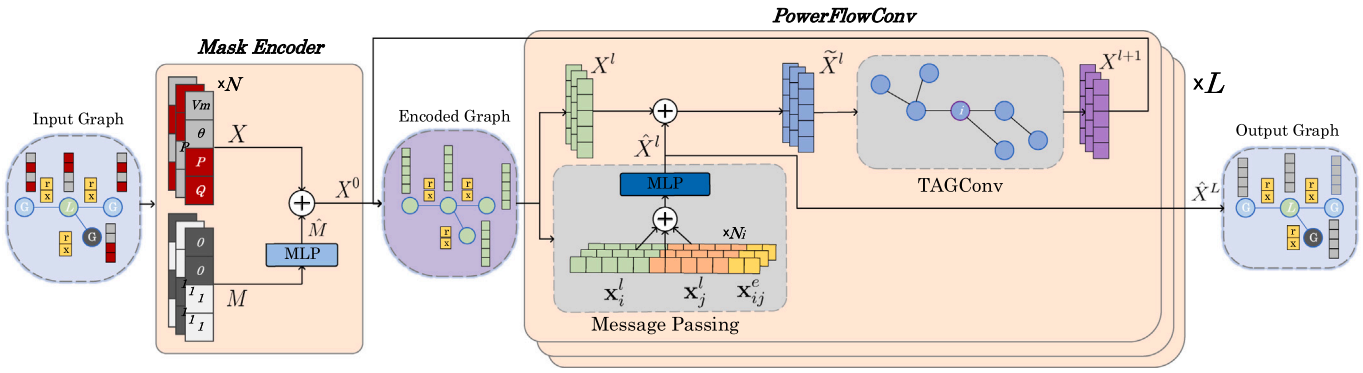


Fig. 3. The PowerFlowNet model architecture consists of a mask encoder and L PowerFlowConv layers. The input graph with incomplete feature information is fed node-by-node to the mask encoder to generate encoded graph features, where each node $n \in \mathcal{N}$ with $(\boldsymbol{x}_i, \boldsymbol{m}_i)$. Then, the encoded graph features are processed by a series of L sequential PowerFlowConv layers, each comprising a 1-step message passing and a high-order TAGConv. Finally, the complete output graph is produced.

$W^l_{MP_1}, b^l_{MP_1}, W^l_{MP_0}, b^l_{MP_0}$ are the trainable parameters of the MLP of layer l.

Afterward, we update the graph features X^l by summing it up with the message $\hat{X}^l$, i.e., $\hat{X}^l \leftarrow X^l + \hat{X}^l$. Then, the signals $\hat{X}^l$ are processed by the TAGConv layer to generate the new encoded graph features X^{l+1} as in

$$X^{l+1} = \sigma\left(\sum_{k=0}^{K-1} (D^{-\frac{1}{2}} A D^{-\frac{1}{2}})^k \hat{X}^l W^l_k\right). \tag{11}$$

In $(D^{-1/2} A D^{-1/2})$, the adjacency matrix A of the graph is normalized by the diagonal degree matrix D. As shown in [22], the normalized adjacency matrix is often used in GNNs to provide computational stability guarantees. Then, the encoded graph features generated by the lth PowerFlowConv layer are passed to the next layer until $l = L$. Finally, the message passing step is used in the last layer while the TAGConv step is discarded, as depicted in Fig. 3.

In general, GNNs are suitable for fast PF approximation due to their nature as function approximators, allowing them to capture complex relationships within the power system efficiently. Moreover, their *permutation equivariant* property [24] ensures that the model processes input data effectively, regardless of the ordering of the buses, enabling parallelization for enhanced computational efficiency. Unlike traditional iterative methods, the proposed approach adopts an end-to-end ML paradigm, where input features are directly mapped to output features in a single step, streamlining the process without requiring iterative convergence.

3.2.3. Model loss functions

The selection of the appropriate model loss function is important in generating high-quality PF approximations. Usually, purely physics-based model losses are used in existing ML approaches for PF [17,18]. However, using only physical model losses is insufficient due to the non-linearities of the PF problem. We propose to utilize the Mean Squared Error (MSE) for the training of PowerFlowNet and develop the Masked L2 Loss as a better evaluation metric. In detail, MSE is defined as

$$\text{MSE}\,(\boldsymbol{y}_i, \hat{\boldsymbol{x}}_i) = \|\boldsymbol{y}_i - \hat{\boldsymbol{x}}_i\|_2^2 \tag{12}$$

where $\boldsymbol{y}_i$ represents the real values of the features and $\hat{\boldsymbol{x}}_i$ are the predicted values of node i.

To get a more precise value of the actual error in predicting only the unknown features, we developed the Masked L2 loss. This loss function is similar to MSE but only calculates the error for the unknown features,

$$\text{MaskedL2}\,(\boldsymbol{y}_i, \hat{\boldsymbol{x}}_i) = \|\boldsymbol{m}_i \circ (\boldsymbol{y}_i - \hat{\boldsymbol{x}}_i)\|_2^2. \tag{13}$$

Here, the $\circ$ is the element-wise multiplication operator, and $\boldsymbol{m}_i$ is the binary mask of node i indexing only the features of interest, as defined in Section 3.2.1.

4. Experimental evaluation

This section presents the experimental setup employed in this study, followed by a comprehensive evaluation aimed at elucidating the strengths and weaknesses of the PowerFlowNet model. To gain deeper insights into the efficacy of PowerFlowNet[4] as a GNN approach for PF approximation, the conducted ablation studies focus on four distinct aspects of the proposed approach: performance, interpretability, scalability, and architectural dependability.

4.1. Experimental setup

The evaluation of PowerFlowNet's performance requires the consideration of the varied attributes present in each power network, encompassing differences in size and topology. To address this requirement, three distinct power network cases are evaluated, namely IEEE case-14, IEEE case-118, and case 6470rte [25], based on their contrasting characteristics. The case with 14 nodes represents a minimal grid topology characterized by a limited number of connections, while the 118 case exhibits a more intricate configuration. Moreover, the 6470rte case accurately emulates the scale and intricacy of the French very high voltage and high voltage transmission network. We evaluate PowerFlowNet in this realistic large-scale network to fully show its scalability potential. Notice that the proposed GNN model is topology dependent; hence, once trained, it cannot be as accurate in unseen cases of topology changes, e.g., N-1 contingency and topology reconfiguration, without any additional training or data samples. While the proposed method may face limitations in generalizing to unseen topologies, there is potential for improvement through training PowerFlowNet with diverse datasets encompassing a spectrum of topology variations. By incorporating multiple datasets that account for N-1 contingencies or offer PF approximations of various grid topologies, PowerFlowNet can enhance its adaptability. Notably, empirical evidence from our scalability study (refer to Section 4.5) underscores the efficacy of PowerFlowNet when trained on heterogeneous datasets. This underscores the significance of selecting an appropriate training dataset tailored to the specific test case, thereby maximizing the effectiveness of our approach.

4.1.1. Dataset generation

All aforementioned cases were modeled in PandaPower [26] and solved using the Newton–Raphson method. For each case, a total of over 30.000 distinct scenarios were generated by perturbing the default case files. Following the same approach as [18], the generation of each

[4] PowerFlowNet code, datasets, and trained models can be found at https://github.com/StavrosOrf/PoweFlowNet and https://github.com/distributionnetworksTUDelft/PoweFlowNet.

sample involved sampling from random uniform or normal distributions. Specifically, the resistance ($\Re[z_{i,j}]$ in Ω) and reactance ($\Im[z_{i,j}]$ in Ω) values of each line were uniformly selected within 80% and 120% of their original values. In the case of generators, the voltage magnitude (V^m) was uniformly set within the range of $[1.00, 1.05]$ per unit, while the initial active powers (P_g in MW) were sampled from a normal distribution $\mathcal{N}(P_g, 0.1|P_g|)$, where the mean corresponds to the original active power and the standard deviation is 10% of it. Lastly, for buses with loads, their active power (P in MW) and reactive power (Q in MVAR) were randomly sampled from normal distributions $\mathcal{N}(P, 0.1|P|)$ and $\mathcal{N}(Q, 0.1|Q|)$, respectively. These samples were then partitioned, allocating 50% for training, 20% for validation, and 30% for testing.

4.1.2. Training & evaluation

To train the models, the AdamW optimizer [27] was employed, using a learning rate of 0.001, in conjunction with the OneCycleLR scheduler [28]. The training process encompassed up to 2000 epochs (for large models) and 1000 epochs (for medium and small networks), with each epoch comprising a batch size of 128 samples. In our standard implementation, we use 4 PowerFlowConv layers ($L = 4$) of order 3 ($K = 3$) to ensure that the computation subgraph of every node covers most of the input graph. PowerFlowNet is developed using PyTorch Geometric [29] components. Additionally, we use 128 nodes for the hidden dimension and a dropout rate of 0.2 at the end of each layer to reduce overfitting. The training time is not considered in the measurement of the execution time since the ML model needs to be trained only once. We train the models with MSE loss as defined in Eq. (12). For assessment purposes, we also trained with the physical loss and the mixed loss ($w = 0.5, \tau = 0.02$), defined in Eqs. (16) and (17) respectively. These additional results can be found in Section 4.5.

Unbalance error. This is a node-wise physical model loss [18], defined by the unbalanced power at each node, i.e. violation of the Kirchhoff's law. This is a self-supervised loss function since it does not require the actual values of every node as presented in Eqs. (14)–(16).

$$\text{Physical}\,(\hat{X}) = \frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}} \|\Delta P_i\|_2^2 + \|\Delta Q_i\|_2^2 \tag{14}$$

$$\Delta P_i = \hat{P}_i - \Re\left[\sum_{j\in\mathcal{N}(i)} \dot{V}_i \left(\frac{\dot{V}_j - \dot{V}_i}{z_{i,j}}\right)^*\right] \tag{15}$$

$$\Delta Q_i = \hat{Q}_i - \Im\left[\sum_{j\in\mathcal{N}(i)} \dot{V}_i \left(\frac{\dot{V}_j - \dot{V}_i}{z_{i,j}}\right)^*\right] \tag{16}$$

The unbalance error model loss function is purely based on the PF equations where $\mathcal{N}(i)$ is the set of neighbor nodes of node i, $\dot{V}_i = \hat{V}_i\angle\hat{\theta}_i$, $(\hat{V}_i, \hat{\theta}_i, \hat{P}_i, \hat{Q}_i)$ is the ith row of $\hat{X}$, and $r_{i,j}$ and $x_{i,j}$ refer to the resistance and reactance of line (i, j), respectively.

Mixed loss. Finally, we propose the Mixed model loss (Eq. (17)) that combines the unbalance error (physical) and MSE loss to efficiently incorporate the PF equations into PowerFlowNet. In detail, the Mixed loss for the whole graph with predicted feature matrix $\hat{X}$ and actual real feature value matrix Y is defined as:

$$\text{Mix}\,(Y, \hat{X}) = w \cdot \text{MSE}\,(Y, \hat{X}) + \tau \cdot (1 - w) \cdot \text{Physical}(\hat{X}) \tag{17}$$

where $w \in [0, 1]$ is a weight to balance the MSE and physical loss, and τ is a scaling factor bringing the physical loss to the same order of magnitude as the MSE loss. In practice, w and τ are hyperparameters that can be tuned during training.

Evaluation. During the evaluation, we compare the Masked L2 loss as a metric.[5] The execution times of each implemented method were examined using high-performance hardware configurations, including the AMD RYZEN 7 5700X 8-Core processor, NVIDIA RTX 3060 TI 8 GB graphics card, and 32 GB RAM, ensuring accurate and efficient measurements. Notably, it is important to emphasize the reproducibility of the experiments, as the availability of datasets, the PowerFlowNet framework, and the trained models provided ensure the feasibility of replicating the study's outcomes and procedures.

4.2. Performance comparison

In this comparison study, PowerFlowNet is evaluated against other baseline methods for PF approximation across three distinct cases, as outlined in Table 2. The performance assessment of each approach is conducted based on accuracy, quantified by employing the Masked L2 Loss, as well as the execution time on the same computing machine. The datasets used for evaluation were generated using the Newton–Raphson method, recognized for its accuracy in PF analysis, which we assume to have negligible error for these particular cases. Note that we did not enforce the generator reactor power limits in the Newton–Raphson solution. As expected, the Newton–Raphson method has increased execution time as the scale of the PF problem grows. The second baseline method considered is the DCPF approach, which relaxes the nonlinear problem to a linear one, thereby serving as an upper-performance boundary for the comparison. However, the DCPF only calculates each bus' voltage angle and active power since it assumes the voltage magnitude to be constant across the grid. PowerFlowNet exhibits significantly lower values for the Masked L2 loss when compared with the DCPF method. Additionally, our proposed approach demonstrates significantly reduced execution time[6] compared to the Newton–Raphson method, with speed improvements of 4× in the 14-node case, 5× in the 118-node case, and 48× in the 6470rte case. Notably, owing to the GNN architecture of PowerFlowNet, the execution time remains consistently low across different cases. This inherent scalability characteristic of GNNs represents a significant advantage over traditional methods, enabling the real-time approximation of PF in extensive power networks. Note that the computation time of Newton–Raphson is acceptable ($\approx$ 0.6 s) when computing a small number of cases but might not be enough when thousands of computations are needed in tasks such as planning where the uncertainty of distributed energy resources is introduced. For example, the Newton–Raphson method would take around 20 minutes to calculate 2000 PF cases for the 6470rte grid, while PowerFlowNet would need less than 1 min.

Furthermore, a comprehensive evaluation of alternative traditional ML approaches was conducted to ascertain the unique benefits of utilizing PowerFlowNet. Initially, the Tikhonov regularizer [30], a method designed to enforce smoothness while considering the underlying graph structure, was assessed. However, as indicated in Table 2, its performance consistently fell behind the other ML methods in terms of loss across all cases, as well as in execution time within the context of large-scale power networks. Subsequently, a three-layer MLP and a simplified GNN consisting of three layers of GCNs [19] were evaluated. Although neither approach surpassed PowerFlowNet in terms of loss (MLP was 1.9× worse and the GCN 17× worse in the 14-case), it is noteworthy that the MLP exhibited a substantially faster execution time, differing by an order of magnitude ($\approx$ 6.5× faster).

In our experiments, the MLPs utilized are intentionally overparameterized, having three fully connected layers, each comprising 128 nodes, with input and output sizes of 4 (representing V^m, θ, P, Q) multiplied by the number of nodes. With approximately 30,000 training samples provided, the combination of a sufficiently large MLP, an ample dataset, and adequate training time leads us to anticipate favorable results. However, while MLPs excel in capturing complex

[5] We also experimented with training with Masked L2 loss, which resulted in almost the same performance as training with MSE loss.

[6] Note that when evaluating NN approaches, the training time is not considered since the model needs to be trained only once.

Table 2
Performance comparison of PowerFlowNet.

Case	14		118		6470rte	
Algorithms	Masked L2 loss	Time (ms)	Masked L2 loss	Time (ms)	Masked L2 loss	Time (ms)
Newton–Raphson	≈0	17.0	≈0	20.0	≈0	580.0
DC power flow	45.74	8.0	99.87	10.0	510.5	30.0
Tikhonov Reg.	2.838	0.4	1.916	0.4	2.934	6100
3-Layer MLP	0.023	0.6	0.175	0.6	0.432	0.6
3-Layer GCN	0.208	1.0	0.591	1.0	1.18	2.0
PowerFlowNet	**0.012**	4.0	**0.018**	4.0	**0.075**	12.0

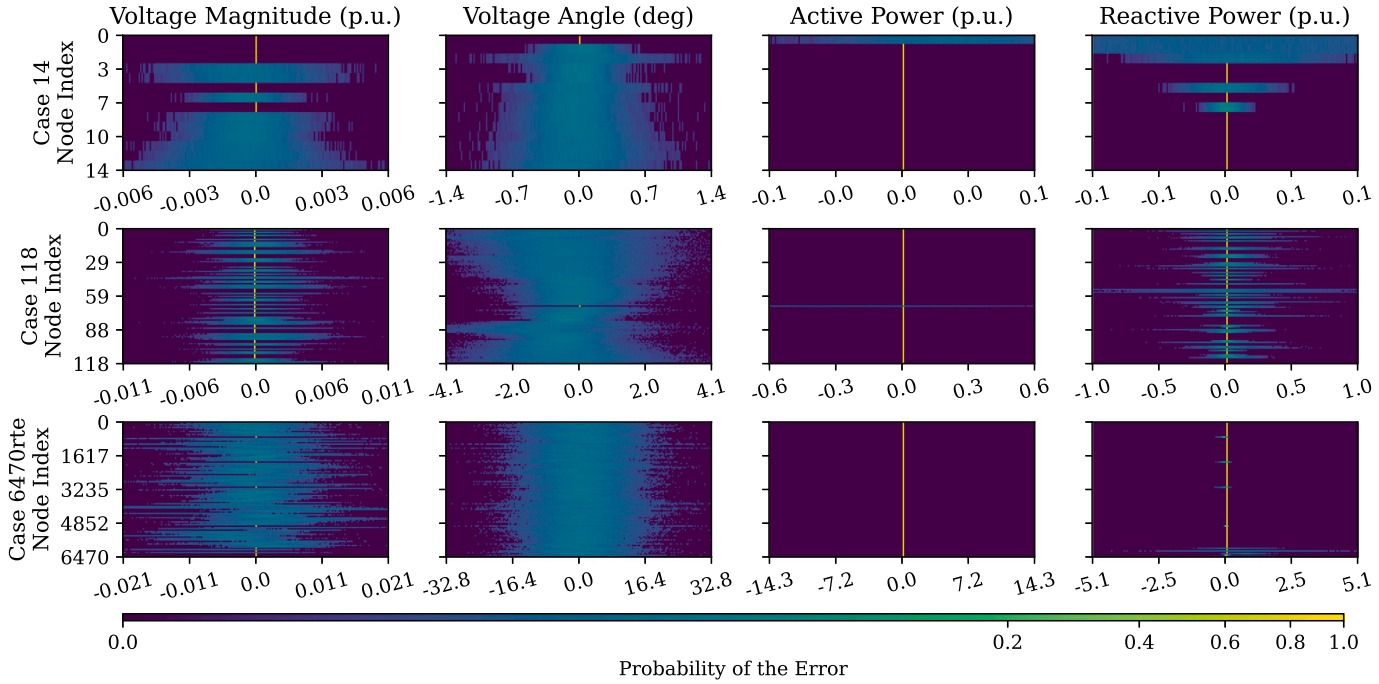


Fig. 4. Probability density function of the actual not-normalized error for every node in the test dataset. The brighter the color, the highest the probability of the prediction error of PowerFlowNet being in that region.

relationships given plenty of resources, they inherently lack the ability to leverage the underlying graph structure of the data. This limitation underscores the importance of GNNs where they show better performance in scenarios with graph-structured data. Despite this, our results unveil that not all types of GNNs perform equally well, as evidenced by the comparative performance against the GCN baseline (Table 2). Thus, the proposed method strategically combines the strengths of both MLPs and GNNs. By integrating features from both architectures, our approach achieves greater effectiveness than the pure GCN or MLP models.

4.3. Denormalized error distributions

After showcasing the performance of PowerFlowNet in three different power networks, it is important to delve deeper into the results and visualize the denormalized error distributions. This analytical approach allows us to gain valuable insights into the areas where more accurate predictions can be achieved, as well as the conditions under which these improvements are most evident. Such insights hold practical significance for power systems operators, enabling them to leverage PowerFlowNet effectively in real-world scenarios involving network operation and strategic planning. Table 3 compares the average error distribution for every one of the four predicted values[7] of PowerFlowNet and the ML baselines (MLP and GCN).

[7] The error distributions include only the predictions of the unknown values.

Table 3
Comparison of the average absolute denormalized error.

	Alg.	V^m (p.u.)	θ (deg)	P (p.u.)	Q (p.u.)
Case 14	MLP	0.0014 ± 0.0011	0.56 ± 0.40	0.04 ± 0.03	0.03 ± 0.03
	GCN	0.0074 ± 0.0053	0.79 ± 0.59	0.16 ± 0.10	0.13 ± 0.12
	Our	0.0011 ± 0.0009	0.20 ± 0.16	0.04 ± 0.03	0.03 ± 0.03
Case 118	MLP	0.0064 ± 0.0049	1.23 ± 1.06	0.54 ± 0.41	0.41 ± 0.34
	GCN	0.0092 ± 0.0068	3.01 ± 2.47	1.79 ± 1.15	0.71 ± 0.77
	Our	0.0012 ± 0.0010	0.87 ± 0.73	0.29 ± 0.21	0.08 ± 0.10
6470rte	MLP	0.0053 ± 0.0046	10.4 ± 7.92	8.91 ± 6.73	0.62 ± 1.75
	GCN	0.0331 ± 0.0273	14.4 ± 12.1	11.2 ± 8.3	0.96 ± 2.11
	Our	0.0045 ± 0.0037	5.49 ± 4.46	7.99 ± 6.01	0.23 ± 0.40

Notice that the proposed model achieves precise predictions of p.u. voltage magnitude for all cases, exhibiting an average error of 11×10^{-4} p.u. in the 14-node scenario and 45×10^{-4} in the notably more intricate 6470rte case, all while maintaining a consistently low standard deviation across each case. In contrast, the MLP and the GCN baselines performed 1.2× and 6.2× worse in the 14-node case and 1.1× and 7.3× in the largest test case, respectively. Additionally, PowerFlowNet's average absolute error remains minimal for both active and reactive power with respect to the network's actual demand and load, with an actual error of 0.04 p.u. for active power and 0.03 p.u. for reactive power in the best-case scenario. Nevertheless, it is important to note a significant rise in the predicted active power error, particularly in the 6470rte case, where it reaches as high as 7.99 (p.u.). Here again, the GCN predicted power results are worse, with

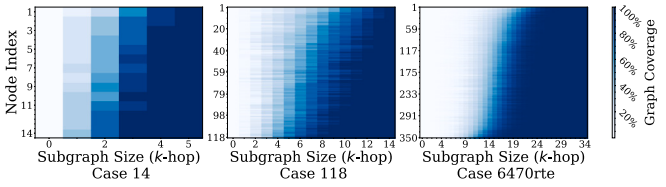


Fig. 5. Illustration of graph coverage defined as the number of nodes included in a subgraph, plotted per K-hop size and sorted from slowest to fastest growth for every node.

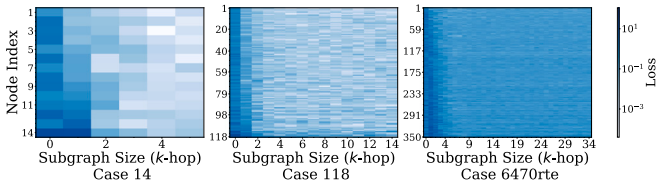


Fig. 6. Masked L2 loss on the central node of a k-hop subgraph for each node and k, sorted by the total node loss.

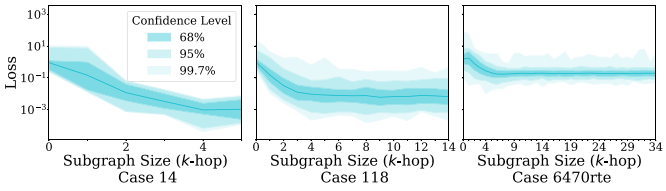


Fig. 7. Average Masked L2 loss on the central nodes of a k-hop subgraph for different numbers of k.

a prediction error of 0.12 p.u. higher in the 14-node case and 3.2 p.u. in the 6470rte case. Notably, the MLP baseline produced equally good power predictions in the 14-node case but gave an average of 0.91 p.u. higher errors for active and 0.39 p.u. for reactive power in the 6470rte case. Furthermore, the voltage angle prediction error for PowerFlowNet is barely noticeable in the 14-node case, being only 0.2 degrees, but increases to 5.49 degrees in the 6470rte case. This occurs because the range of the voltage angles across larger networks network, i.e., the 6470rte case, is much greater. Similarly, high-angle errors persist with the ML baselines too. In detail, GCN provided angle predictions with a mean of 14.4 degrees of error and 10.4 degrees error for the MLP in the largest node case.

Moreover, we can observe the per-node denormalized error distribution for every case in Fig. 4. Notice that the prediction error for nodes where we already know the value is practically zero (indicated by the vertical yellow line). Although the average voltage angle error is higher for the 6470rte case (5.49 degrees as shown in Table 3), the average voltage magnitude error is in the same range (less than 0.0045 p.u.) as in the smaller networks, ensuring that the developed PowerFlowNet is accurate enough for fast checking of operational constraints for large numbers of scenarios regardless of network size.

4.4. Sensitivity to the hop size

To gain insights into the locality of information used by PowerFlowNet to make predictions, we aimed to investigate the potential of using subgraphs as a more scalable and generalizable approach to tackle the challenges posed by the complex PF prediction task.

In this study, we define the receptive field as the longest distance between any two nodes that can exchange information. Naturally, a larger K and L would increase the receptive field. We examined K-hop subgraphs across all three cases, where, for reference, a PowerFlowNet model with $K = 3$ and four layers of TAGConv ($L = 4$), a straightforward computation yields an effective receptive field spanning 12 nodes. In practice, the size of these receptive fields, measured in the number of nodes, will vary based on the specific graph topology. To better understand this topological effect, we looked at the speed at which K-hop subgraphs grow from each node in the graph, also defined as graph coverage. In detail, for each node, we made K-hop subgraphs for K between 1 and the graph diameter and measured the subgraph size after each hop, as shown in Fig. 5. The graph diameter is the longest distance between any pair of nodes within a graph, representing the maximum distance between any two points in the network. This indicates the distribution of receptive field sizes seen during training; interestingly, it appears to form a Gaussian shape, particularly visible in the largest graph in the dataset with 6470 nodes. As expected, the larger the graph size the more hops are required to cover the whole graph. For example, 4 hops are required in the 14-node over 6 hops in the 118-node case, and 11 to 25 hops in the 6470rte case.

Afterward, those subgraphs were each passed through the PowerFlowNet model, and the Masked L2 loss was calculated only on the node from which the K-hop neighborhood was generated. This showed us the actual loss distribution among nodes depending on how far a K-hop neighborhood can reach. The detailed results are illustrated in Fig. 6.

We noticed that this loss distribution has a different shape from the subgraph growth figure, and indeed there is variation in which nodes are more or less affected by a decrease in subgraph size.

Interestingly, we can observe that only around 3 hops are required to obtain the minimum loss for most of the nodes, experimentally proving the reason TAGConv of order $K = 3$ can achieve high-quality results. This also suggests that for most buses (nodes) in a power network, we could make PowerFlowNet predict the PF values even faster by only looking at smaller connected parts.

Finally, we plot the average loss across nodes in Fig. 7 and discover that neighborhood hop counts well below the effective receptive field's hop count are enough to make predictions practically identical to those when using the full graph. This indicates that local predictions in a large network are feasible. Most importantly, we argue that training on larger graphs could be made more efficiently by sampling subgraphs across different graphs available in the dataset, so as to make sure that stochastic gradient descent steps are performed using batches that capture more of the variation available in the dataset. Subsequently, this approach has the potential to augment the scalability aspects of PowerFlowNet.

4.5. Network generalization capabilities

We demonstrate that our model has valuable generalization features to perform well in different network topologies after training in one particular topology. Also, we investigate how PowerFlowNet's performance scales with the NN's model size, the size of the training dataset, and the model loss function used.

Based on the standard setting PowerFlowNet (Medium) with $L = 4, h = 128$, we created two variants by modifying the total number of the model's trainable parameters by one order of magnitude as shown in Table 4. In detail, we created PowerFlowNet (Small) with $L = 2, h = 64$ and PowerFlowNet (Large) with $L = 5, h = 512$. Table 4 shows the relation between the model size and the Masked L2 loss when trained with different model loss functions for the 118-node case. We note that the performance scales very well with the model size. Most importantly, from the medium to large model, the Masked L2 loss is almost one order of magnitude lower, signifying the learning potential of PowerFlowNet.

The model performance does not differ greatly when trained with physical loss (as used in [18]). This means learning with physical loss is challenging regardless of the model's capability. However, we notice that the Masked L2 performance difference between training with MSE and mixed loss shrinks as we move to the large model. This suggests

Table 4
Masked L2 loss of scaled models on the 118 case.

Model	# Params	Trained with		
		MSE	Physical	Mixed
PowerFlowNet (Small)	32k	0.079	0.757	0.109
PowerFlowNet	357k	0.018	0.667	0.057
PowerFlowNet (Large)	7375k	**0.002**	**0.628**	**0.004**

Table 5
Performance evaluation for varying training cases.

Eval. Case		118			14		
Eval. Metric		M. L2	MSE	Phys.	M. L2	MSE	Phys.
Train. Case	118+14 (L)	**0.0023**	**0.0011**	**4.9676**	**0.0014**	**0.0006**	**0.1336**
	118 (L)	0.0029	0.0014	5.4080	6.1998	2.5947	40.942
	118 (M)	0.0182	0.0105	58.9221	3.7860	3.7860	45.426

that this capability gain allows us to optimize both the MSE and the physical loss, thus making precise and physics-conforming predictions.

To further test PowerFlowNet's capability, we trained the large model on a mixed dataset composed of the 14 and the 118 cases (118 + 14). We unified the training loss function with MSE loss and evaluated the trained models using different metrics (MSE, Masked L2, and physical loss) on each of the two cases individually. The results are given in Table 5. After comparing with training large (L) and medium (M) models on the 118-node case, we noticed that even without explicitly bringing physical model loss in training, the large models achieve better performance. Large models are around 10× better in terms of unbalance error when trained and evaluated on the 118-node case. When we augment the 118-node case training set with the 14-node case, the Large PowerFlowNet model performs better than when trained only in the 118-node case achieving an excelling accuracy in the 14-node case . This suggests that the proposed large PowerFlowNet model can operate on multiple graphs and learn the underlying physics without sacrificing accuracy.

4.6. Extreme PF scenarios evaluation

We conducted generalization experiments to assess PowerFlowNet's performance on datasets with higher standard deviations, representing out-of-distribution scenarios. All models were trained on the dataset described in Section 4.1.1. To create out-of-distribution data, we sampled the power setpoints of generators, P_g from a normal distribution $\mathcal{N}(P_g, \sigma \cdot |P_g|)$. Similarly, we sampled the active power P for load buses from $\mathcal{N}(P, \sigma \cdot |P|)$. Here, σ varies between 0.05 and 0.5, providing a range of test conditions. A standard deviation of $\sigma = 0.1$ in the training dataset represents *normal* PF scenarios, while $\sigma = 0.5$ represents more extreme cases, challenging the model's robustness and generalization capabilities.

Fig. 8 illustrates the evaluation results on out-of-distribution data in terms of root mean square error (RMSE) of V^m (p.u) and θ (deg.). RMSE is defined as:

$$\text{RMSE}(\boldsymbol{y}_i, \hat{\boldsymbol{x}}_i) = \sqrt{\frac{\sum_{i=1}^{N} \left(\boldsymbol{y}_i - \hat{\boldsymbol{x}}_i\right)^2}{N}}, \tag{18}$$

where $\boldsymbol{y}_i$ represents the real values of the features and $\hat{\boldsymbol{x}}_i$ are the predicted values of node i. As anticipated, the error generally rises linearly with an increase in the test set's standard deviation. It is critical for ML models to have training data distributions closely aligned with test data distributions to achieve accurate results [31]. In the 14-node case (see Fig. 8(a)), the error in predicted voltage magnitude (V^m) increases up to 0.0035 p.u., while the angle error can be up to 1.2 degrees higher. Similarly, for the 118-node case (Fig. 8(b)), the out-of-distribution error can reach up to 0.004 p.u. for voltage magnitude and as high as 10 degrees for angle predictions. Finally, in the more

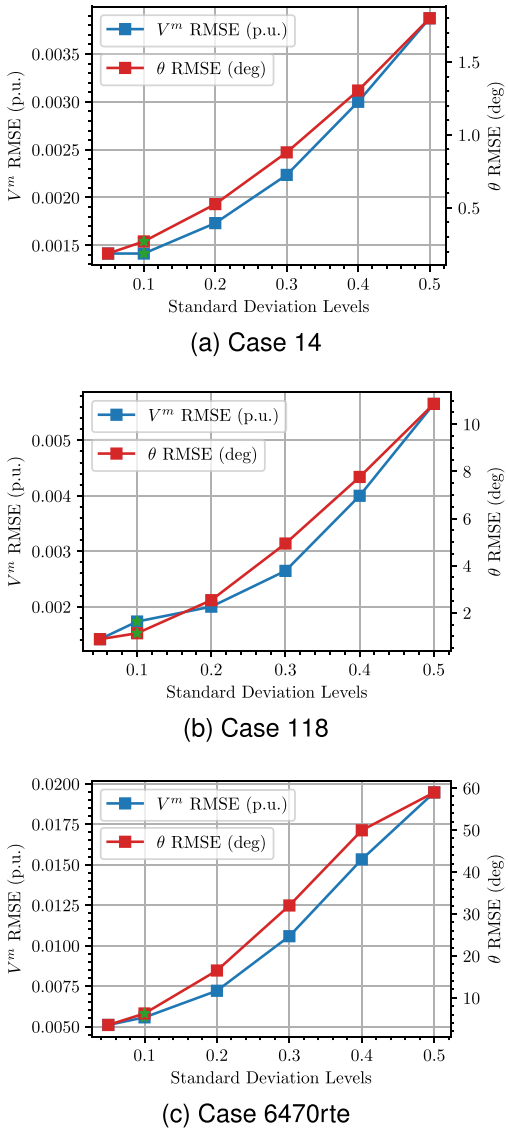


Fig. 8. Out-of-distribution error of voltage magnitude and angle. The model is trained with 0.1 standard deviation (indicated by a star).

complex 6470rte case (Fig. 8(c)), angle errors can escalate sharply, with deviations reaching up to 60 degrees when $\sigma = 0.5$. This significant error increase is likely due to the complexity and scale of the power network.

Testing with out-of-distribution data presents a substantial challenge in ML [32]. Despite this, PowerFlowNet has demonstrated solid generalization performance in the 14-node and 118-node cases, even under extreme PF scenarios. To further improve robustness, the training dataset could be augmented with additional extreme PF scenarios, allowing PowerFlowNet to better learn and adapt to these challenging situations.

In practice, PF calculations often encounter PV/PQ switching scenarios, where the reactive power limit of a PV bus is reached and converted to a PQ bus. This paper only focuses on the most general scenarios without including the PV/PQ switching in the training. However, our model naturally accommodates PV/PQ switching due to the GNN architecture and mask encoder. In Table 6, we evaluate our model's performance when encountering PV/PQ switching. Even when the model has never seen PV/PQ switching in training, PowerFlowNet can generalize and give reasonable voltage predictions. Notably, if

Table 6
MAE of standard and PV/PQ switching scenarios in 118 case.

Scenario	V^m (p.u.)	θ (deg)	P (p.u.)	Q (p.u.)
Standard 118 case	0.0012	0.8600	0.2851	0.0839
PV/PQ switching 118 case	0.0149	6.4485	2.6290	0.3387

Table 7
PowerFlowNet component significance analysis.

Case	Full model	Model 1-Layer	No message passing (MP)	1-Layer & No MP
14	**0.012**	0.04	0.08	0.38
118	**0.018**	0.21	0.22	0.95
6470rte	**0.075**	1.01	1.41	6.11

addressing PV/PQ switching properly with higher precision is important in one's application, additional PV/PQ switching training samples should be included in the training.

4.7. Architectural ablation study

Finally, architectural ablation experiments were carried out to demonstrate the vitality of PowerFlowNet's unique structures through component removal. Therefore, the experiments were designed to highlight how each PowerFlowNet component contributes to generating high-quality predictions. Table 7 depicts the results of these experiments. Table 7 is split into four variations of PowerFlowNet: the full model, PowerFlowNet with $L = 1$, PowerFlowNet without message passing, and PowerFlowNet with $L = 1$ and without message passing. Then, we can observe how the model performs in terms of Masked L2 loss in all different scenarios.

As observed, none of the PowerFlowNet's variations performed as well as the complete model. In the 1-layer case without message passing, it is visible that the model fails to capture distant node dependencies, resulting in up to 65× higher error. Similarly, the model performs poorly (up to 6.6× higher error) in the no message passing case since the edge features are completely ignored, highlighting the importance of incorporating the edge features. On the other hand, the 1-layer model performs the best (up to 3.3× higher error). Consequently, this analysis shows that every part of PowerFlowNet is important in achieving high-quality PF approximations.

5. Conclusions

In this paper, we presented PowerFlowNet, a novel data-driven algorithm that capitalized on the efficiency and capabilities of GNN operations applied to the power network's topology, leading to a accurate approximation of the PF. Specifically, our approach transformed the traditional PF problem into a GNN node-regression task by representing buses as nodes and transmission lines as edges while preserving network connectivity. Our model featured a distinctive configuration involving a mask encoder in conjunction with a sequence of our proposed PowerFlowConv layers, designed to aggregate features across the entire graph and inherently learn the dynamics of the underlying PF. The results verified PowerFlowNet's ability to generate high-quality solutions for a diverse set of network cases including the 6470rte very large-scale power network of France. Most importantly, our method produces high-quality PF predictions at a fraction of the time of traditional solvers, regardless of the scale of the network, outperforming the well-established Newton–Raphson method in terms of execution time while having closely comparable results. Ultimately, our ablation studies reveal the robust aspects and potential limitations of our approach, thereby enabling the versatile deployment of PowerFlowNet across a wide array of real-world power system operation and planning scenarios.

CRediT authorship contribution statement

Nan Lin: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **Stavros Orfanoudakis:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Data curation, Conceptualization. **Nathan Ordonez Cardenas:** Methodology, Investigation, Conceptualization. **Juan S. Giraldo:** Writing – review & editing. **Pedro P. Vergara:** Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

This manuscript has not been published, is not under consideration for publication elsewhere, and we have no conflicts of interest to disclose.

Data availability

Link to the data is provided in the paper.

Acknowledgments

This work used the Dutch national e-infrastructure with the support of the SURF Cooperative, The Netherlands using grant no. EINF-6262. This publication is part of the project ALIGN4energy (with project number NWA.1389.20.251) of the research program NWA ORC 2020, which is (partly) financed by the Dutch Research Council (NWO), The Netherlands. Stavros is funded by the HORIZON Europe Drive2X Project 101056934.

References

[1] Albadi M. Power flow analysis. In: Volkov K, editor. Computational models in engineering. Rijeka: IntechOpen; 2019, http://dx.doi.org/10.5772/intechopen.83374, Ch. 5.
[2] da Costa VM, Martins N, Pereira JLR. Developments in the Newton Raphson power flow formulation based on current injections. IEEE Trans Power Syst 1999;14:1320–6.
[3] Eltamaly A, A. Elghaffar A. Load flow analysis by Gauss-Seidel method; a survey. Int J Mechatron Electr Comput Technol (IJMEC) 2017;x. 2305-0543, 2411-6173.
[4] Li X, Li F, Yuan H, Cui H, Hu Q. GPU-based fast decoupled power flow with preconditioned iterative solver and inexact Newton method. IEEE Trans Power Syst 2017;32(4):2695–703. http://dx.doi.org/10.1109/TPWRS.2016.2618889.
[5] Seifi H, Sepasian M. Electric power system planning: Issues, algorithms and solutions, vol. 49, Springer; 2011, http://dx.doi.org/10.1007/978-3-642-17989-1.
[6] Liu Y, Zhang N, Wang Y, Yang J, Kang C. Data-driven power flow linearization: A regression approach. IEEE Trans Smart Grid 2019;10(3):2569–80. http://dx.doi.org/10.1109/TSG.2018.2805169.
[7] Guo L, Zhang Y, Li X, Wang Z, Liu Y, Bai L, Wang C. Data-driven power flow calculation method: A lifting dimension linear regression approach. IEEE Trans Power Syst 2022;37(3):1798–808. http://dx.doi.org/10.1109/TPWRS.2021.3112461.
[8] Chen J, Wu W, Roald LA. Data-driven piecewise linearization for distribution three-phase stochastic power flow. IEEE Trans Smart Grid 2022;13(2):1035–48. http://dx.doi.org/10.1109/TSG.2021.3137863.
[9] Chen Y, Wu C, Qi J. Data-driven power flow method based on exact linear regression equations. J Mod Power Syst Clean Energy 2022;10(3):800–4. http://dx.doi.org/10.35833/MPCE.2020.000738.
[10] Liu Y, Wang Y, Zhang N, Lu D, Kang C. A data-driven approach to linearize power flow equations considering measurement noise. IEEE Trans Smart Grid 2020;11(3):2576–87. http://dx.doi.org/10.1109/TSG.2019.2957799.
[11] Ramasamy S, Ganesan K, Arunachalam V. Power flow parameter estimation in power system using machine learning techniques under varying load conditions. Int J Electr Electron Res 2022;10(4):1299–305. http://dx.doi.org/10.37391/IJEER.100484.
[12] Hu X, Hu H, Verma S, Zhang Z-L. Physics-guided deep neural networks for power flow analysis. IEEE Trans Power Syst 2021;36(3):2082–92. http://dx.doi.org/10.1109/TPWRS.2020.3029557.
[13] Yang Y, Yang Z, Yu J, Zhang B, Zhang Y, Yu H. Fast calculation of probabilistic power flow: A model-based deep learning approach. IEEE Trans Smart Grid 2020;11(3):2235–44. http://dx.doi.org/10.1109/TSG.2019.2950115.

[14] Li H, Weng Y, Vittal V, Blasch E. Distribution grid topology and parameter estimation using deep-shallow neural network with physical consistency. IEEE Trans Smart Grid 2023;1. http://dx.doi.org/10.1109/TSG.2023.3278702.
[15] Yang Q, Sadeghi A, Wang G, Giannakis GB, Sun J. Power system state estimation using Gauss-Newton unrolled neural networks with trainable priors. In: 2020 IEEE smartgridcomm. 2020, p. 1–6.
[16] Wu T, Carreno IL, Scaglione A, Arnold D. Spatio-temporal graph convolutional neural networks for physics-aware grid learning algorithms. IEEE Trans Smart Grid 2023. http://dx.doi.org/10.1109/TSG.2023.3239740.
[17] Habib B, Isufi E, Breda Wv, Jongepier A, Cremer JL. Deep statistical solver for distribution system state estimation. IEEE Trans Power Syst 2023;1–12. http://dx.doi.org/10.1109/TPWRS.2023.3290358.
[18] Donon B, Clément R, Donnot B, Marot A, Guyon I, Schoenauer M. Neural networks for power flow: Graph neural solver. Electr Power Syst Res 2020;189:106547. http://dx.doi.org/10.1016/j.epsr.2020.106547.
[19] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: 5th international conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, conference track proceedings. OpenReview.net; 2017.
[20] Liao W, Bak-Jensen B, Pillai JR, Wang Y, Wang Y. A review of graph neural networks and their applications in power systems. J Mod Power Syst Clean Energy 2022;10(2):345–60. http://dx.doi.org/10.35833/MPCE.2021.000058.
[21] Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. In: International conference on machine learning. PMLR; 2017, p. 1263–72.
[22] Du J, Zhang S, Wu G, Moura JM, Kar S. Topology adaptive graph convolutional networks. 2017, arXiv preprint arXiv:1710.10370.
[23] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: 5th international conference on learning representations, ICLR 2017 - conference track proceedings. International Conference on Learning Representations, ICLR; 2016, URL https://arxiv.org/abs/1609.02907v4.
[24] Gama F, Isufi E, Leus G, Ribeiro A. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. IEEE Signal Process Mag 2020;37(6):128–38. http://dx.doi.org/10.1109/MSP.2020.3016143.
[25] Josz C, Fliscounakis S, Maeght J, Panciatici P. AC power flow data in MATPOWER and QCQP format: iTesla, RTE snapshots, and PEGASE. 2016, arXiv: Optimization and Control.
[26] Thurner L, Scheidler A, Schäfer F, Menke J, Dollichon J, Meier F, Meinecke S, Braun M. Pandapower — An open-source python tool for convenient modeling, analysis, and optimization of electric power systems. IEEE Trans Power Syst 2018;33(6):6510–21. http://dx.doi.org/10.1109/TPWRS.2018.2829021.
[27] Loshchilov I, Hutter F. Decoupled weight decay regularization. In: International conference on learning representations. 2017.
[28] Smith LN, Topin N. Super-convergence: very fast training of neural networks using large learning rates. In: Defense + commercial sensing. 2017.
[29] Fey M, Lenssen JE. Fast graph representation learning with PyTorch Geometric. In: ICLR workshop on representation learning on graphs and manifolds. 2019.
[30] Hilt DE, Seegrist DW, Service. USF, Station NFE. Ridge, a computer program for calculating ridge regression estimates. Vol. no.236, Upper Darby, Pa, Dept. of Agriculture, Forest Service, Northeastern Forest Experiment Station; 1977, p. 10.
[31] Mohseni S, Pitale M, Yadawa J, Wang Z. Self-supervised learning for generalizable out-of-distribution detection. In: Proceedings of the AAAI conference on artificial intelligence. Vol. 34, 2020, p. 5216–23. http://dx.doi.org/10.1609/aaai.v34i04.5966, (04). URL https://ojs.aaai.org/index.php/AAAI/article/view/5966.
[32] Yang J, Zhou K, Li Y, Liu Z. Generalized out-of-distribution detection: A survey. 2024, arXiv:2110.11334.