

Power Failure Cascade Prediction using Graph Neural Networks

Sathwik Chadaga, Xinyu Wu, and Eytan Modiano

Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA

Abstract—We consider the problem of predicting power failure cascades due to branch failures. We propose a flow-free model based on graph neural networks that predicts grid states at every generation of a cascade process given an initial contingency and power injection values. We train the proposed model using a cascade sequence data pool generated from simulations. We then evaluate our model at various levels of granularity. We present several error metrics that gauge the model’s ability to predict the failure size, the final grid state, and the failure time steps of each branch within the cascade. We benchmark the graph neural network model against influence models. We show that, in addition to being generic over randomly scaled power injection values, the graph neural network model outperforms multiple influence models that are built specifically for their corresponding loading profiles. Finally, we show that the proposed model reduces the computational time by almost two orders of magnitude.

Index Terms—Power failure cascade, contingency analysis, graph neural networks.

I. INTRODUCTION

Modern power grids often experience unpredictable component failures that are caused due to an exogenous event like a tree branch falling, bad weather, failure of an aged device, or an operator error. These random failures, if not treated properly, can propagate rapidly through the grid, potentially resulting in large scale blackouts. Hence, it is important to study such *failure cascades* as part of the power contingency analysis. Moreover, power grids have seen a recent surge in outages [1] due to extreme weather conditions [2] and power grid aging [3], causing significant losses to businesses, industries, and healthcare sectors [4], [5], making the study of power failures increasingly important.

There have been several studies performed on historical failure cascade data [6], [7]. However, the scarce historical records of cascading failures are not representative of all the possibilities. Hence, numerical simulations and analysis methods have been proposed, which solve the static power flow problem step-by-step and determine the sequence of quasi-static transmission link overflows [8]. However, the AC power flow model is computationally expensive, while the computationally tractable DC power flow model has been shown to underestimate the failure sizes [9].

To overcome the high complexity of flow-based methods, efforts have been devoted to constructing flow-free models of failure cascades. Tools like branching process [6], the random chemistry algorithm [10], and the expectation-maximization algorithm [11] have been proposed to estimate blackout risks. These flow-free models aspire to capture the cascade flow

dynamics from data, obtained either from simulations or historic outage records. This data driven approach has led researchers to investigate fast and accurate machine learning models.

Machine learning has been used in power system analysis in various settings [12]. For example, as power flow calculation using Newton-Raphson is computationally expensive, more efficient power flow calculation methods have been proposed using deep [13] and convolutional [14] neural networks. Moreover, in the area of cascade prediction, support vector machines have been employed in blackout prediction [15], cascade failure size estimation [16], and load loss estimation [17]. Additionally, methods using Bayes networks have been proposed for failure cascade prediction in [18]. Despite being computationally efficient, these techniques fail to take advantage of the power grid topology information leading us to explore techniques that use graph neural networks.

Graph neural networks (GNNs) are a type of neural networks that operate on graph-structured data [19]. They process input graphs by repetitively updating the information at each node based on its neighbors, thereby leveraging the underlying graph topology. There have been recent applications of GNNs in the field of power networks. One such application is the design of computationally efficient power flow solvers. In [20]–[22], GNNs are trained in a supervised way to imitate the Newton-Raphson power flow solver. Whereas [23], [24] follow an unsupervised learning method that minimizes the violation of Kirchoff’s laws.

Moreover, GNNs have seen recent applications in the field of power failure cascades. GNNs have been used for real time grid monitoring tasks during a cascade, like predictions of optimal load shedding [25] and total load lost [26]. These works involve a graph-level prediction task, i.e. they predict a particular property of the grid as a whole. GNNs can also be used for edge-level and node-level prediction tasks. For example, in [27], a node-level vulnerability metric called the Avalanche Centrality is predicted for all nodes of the grid using GNNs. In [28], efficient failure cascade path search techniques with GNNs have been proposed.

The existing works as discussed above are focused on characterizing one or two aspects of failure cascades, like load loss, failure size, or blackout possibility, lacking a comprehensive evaluation of the cascade at finer levels of granularity. This is addressed in [29], where an influence model is trained to predict the power grid states within a cascade. However, the influence model approach cannot generalize for variable

loading as it does not take as input the power injection values. A flow-based GNN model has been proposed in [30] that can generalize for variable power injections. However, this model is centered around predicting the power flow values in a step-by-step manner to obtain the sequence of branch overflows. Hence, even though this technique speeds up the cascade prediction process compared to traditional methods, it still involves a high computational overhead in handling the formation of islands during the cascade, such as identification of islands and rebalancing the load and power generation within islands.

In this paper, we build a flow-free GNN model that does cascade sequence prediction without requiring power flow calculation at every generation of the cascade. We summarize our contributions below.

- 1) We propose a flow-free model based on a GNN that predicts grid states at every generation of a cascade, providing a way to comprehensively evaluate cascades at various levels of granularity. The proposed model takes as input the node power injection values, the initial contingency, and the grid topology. We use the cascading failure simulator oracle from [10] to generate a cascade sequence dataset to train our model.
- 2) We evaluate the performance of our model at various levels of granularity including prediction of the failure size, the final grid state, and the generations at which each branch fails within a cascade. We benchmark our model against the influence model [29] and show that in addition to being generic over randomly scaled loading values, the GNN model outperforms different load-specific influence models under every metric.
- 3) We perform a runtime analysis and show that the GNN model reduces the prediction time by almost two orders of magnitude compared to the DC power flow calculation based simulators.

The rest of the paper is organized as follows. We formulate the problem of failure cascade prediction and describe the proposed graph neural network model in Section II. We discuss the cascading failure simulator oracle in Section III. We present the model performance results in Section IV.

II. PROBLEM FORMULATION AND THE GNN MODEL

We consider the power failure cascade process due to branch failures. In this setting, a failure cascade begins with an initial failure of one or more branches in the grid. The initial branch failures perturb the power flow in the grid, leading other branches to overload and trip. The new failures further cause additional branches to trip and so on, consequently triggering a cascade process. The cascade process can be grouped into generations in time [6], which we refer to as time steps.

We represent the power grid by a directed graph $G = (V, E)$, where the nodes V represent buses and the directed edges E represent branches (or edges). For a branch $e \in E$ at time t , we choose the *branch state* $s_e[t]$ to be its binary operational state, which can either be 0 (failed) or 1 (active). We define the *network state* at time t as $s[t] := (s_e[t])_{e \in E}$.

Given the *initial contingency* $s[0]$ (the network state at $t = 0$), our goal is to predict the *cascade sequence* $s := (s[t])_{t=0}^{T-1}$, where T is the cascade length. However, we assume that once a branch fails, it stays in the failed state for the rest of the cascade. This allows us to define the *failure step* of a branch e , the time step at which its state changes from 1 to 0, as $f_e := \sum_{t=0}^{T-1} s_e[t]$. From this failure step f_e , we can fully recover the branch states $s_e[t]$, and hence s , by setting $s_e[t] = 1$ for $0 \leq t < f_e$ and $s_e[t] = 0$ for $f_e \leq t < T$ for all $e \in E$. Hence, instead of predicting s directly, we design a model that predicts the branch failure steps $f := (f_e)_{e \in E}$. We propose a GNN model to do this as explained below.

The proposed model takes as input the topology of the grid $G = (V, E)$, the initial contingency $s[0] \in \{0, 1\}^{|E|}$, and the power injection values $P_v \in \mathbb{R}$ at each node $v \in V$. In this model, we process the input data in multiple stages as explained in the following paragraphs. Fig. 1 shows a block diagram that summarizes the model.

1) *Initial Stage*: We start by removing the edges corresponding to failed branches in the initial contingency and get the new set of edges $E' = \{e \in E : s_e[0] = 1\}$. Then, we pass the node power injection values P_v through a neural network to obtain the transformed node features $\tilde{P}_v \in \mathbb{R}^L$ as follows.

$$\forall v \in V, \quad \tilde{P}_v = H_{\text{initial}}(P_v) \quad (1)$$

where, the mapping $H_{\text{initial}} : \mathbb{R} \rightarrow \mathbb{R}^L$ represents a dense neural network whose weights will be learned through back propagation during the training phase. Note that the same neural network is being used on all the nodes. Further, we use these values \tilde{P}_v to initiate the edge hidden features h_e^0 as

$$\forall e = (u, v) \in E', \quad h_e^0 = h_{(u,v)}^0 = \tilde{P}_u - \tilde{P}_v \quad (2)$$

where, a directed edge $e \in E'$ is represented as $e = (u, v)$ with $u, v \in V$ being its source and destination nodes respectively.

2) *Attention Stage*: For an edge $e = (u, v) \in E'$, we define the set of adjacent edges as $\mathcal{N}_e = \mathcal{N}_{(u,v)} = \{(u, w) : w \in V, (u, w) \in E'\} \cup \{(w, v) : w \in V, (w, v) \in E'\}$. Now, we generate two types of attention coefficients: edge-to-edge a_{ed} for every two neighboring edges $e \in E', d \in \mathcal{N}_e$; and node-to-edge (b_{eu}, b_{ev}) for all edges $e = (u, v) \in E'$ and their nodes u, v . We generate these coefficients by passing the given initial contingency $s[0]$ through two dense neural networks as

$$a = H_{\text{edge-edge}}^{\text{attn}}(s[0]), \quad b = H_{\text{node-edge}}^{\text{attn}}(s[0]) \quad (3)$$

where, the mappings $H_{\text{edge-edge}}^{\text{attn}} : \mathbb{R}^{|E|} \rightarrow \mathbb{R}^{\sum_{e \in E'} (|\mathcal{N}_e| - 1)}$ and $H_{\text{node-edge}}^{\text{attn}} : \mathbb{R}^{|E|} \rightarrow \mathbb{R}^{2|E|}$ represent two dense neural networks whose weights will be learned through back propagation during training, a is the collection of edge-to-edge coefficients $a = (a_{ed})_{\{e, d \in E' : d \in \mathcal{N}_e, d \neq e\}}$, and b is the collection of node-to-edge coefficients $b = (b_{eu}, b_{ev})_{e=(u,v) \in E'}$.

These attention coefficients will be used in the next averaging stages, where we repetitively update edge hidden features by weighted-averages of neighboring edge and node hidden features. The attention coefficients will act as weights for this purpose. They represent how much weight, or attention, needs

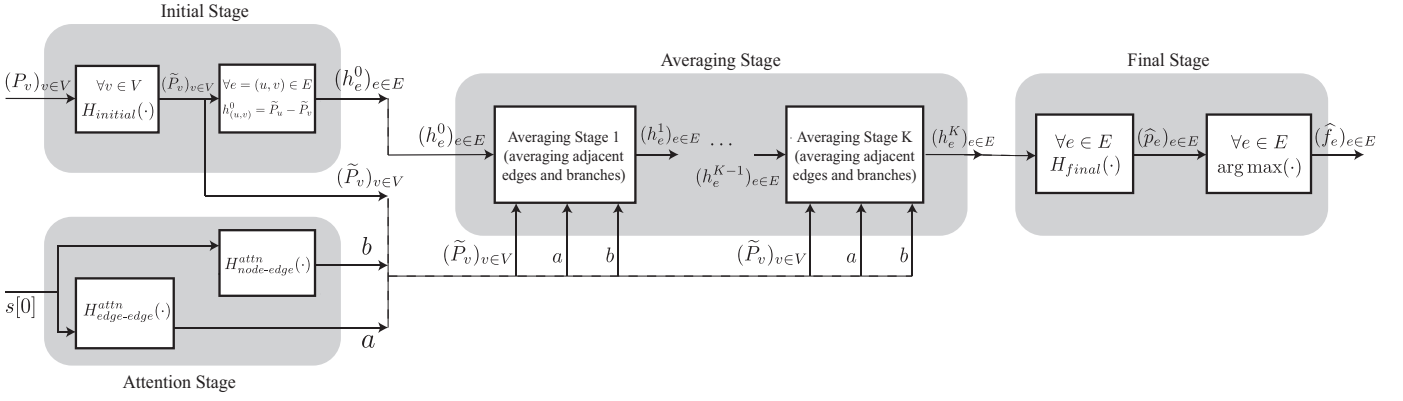


Fig. 1: Block diagram of the GNN model.

to be given on adjacent edges and nodes while updating an edge's hidden features.

3) *Averaging Stage*: In this stage, we pass the outputs of the initial stage h_e^0 through a sequence of K averaging steps. In each step $k = 1, \dots, K$, we calculate the weighted average of neighboring edge and node features and pass them through a neural network to obtain the new edge features as follows.

$$\forall k = 1, \dots, K, \forall e = (u, v) \in E',$$

$$h_e^k = \frac{h_e^{k-1}}{|\mathcal{N}_e|} + H_{edge-edge}^k \left(\sum_{d \in \mathcal{N}_e, d \neq e} \frac{a_{ed}}{\sqrt{|\mathcal{N}_e|} \sqrt{|\mathcal{N}_d|}} h_d^{k-1} \right) + H_{node-edge}^k \left(\frac{b_{eu} \tilde{P}_u + b_{ev} \tilde{P}_v}{2} \right) \quad (4)$$

where, K is the total number of averaging steps, h_e^k is the edge hidden feature of edge e at k -th averaging step, and \mathcal{N}_e is the set of edges that are adjacent to and including e . The coefficients a_{ed} and b_{ue} are the edge-to-edge and node-to-edge attention coefficients respectively obtained in the attention stage. Finally, the functions $H_{edge-edge}^k, H_{node-edge}^k : \mathbb{R}^L \rightarrow \mathbb{R}^L$ for $k = 1, \dots, K$ represent dense neural networks, whose weights will be learned through back propagation during the training phase. Note that these neural networks are the same for all edges.

4) *Final Stage*: In the final stage, we predict the failure step probability values $\hat{p}_e := [\hat{p}_{e,0}, \dots, \hat{p}_{e,T-1}]$ for all branches $e \in E'$, where each entry $\hat{p}_{e,t}$ is the predicted probability that branch e fails at time t . We do this by passing the output of last averaging step h_e^K through a dense neural network as

$$\forall e \in E', \quad \hat{p}_e = H_{final}(h_e^K) \in [0, 1]^T \quad (5)$$

where, the function $H_{final} : \mathbb{R}^L \rightarrow \mathbb{R}^T$ represents a dense neural network, whose weights will be learned during the training phase. This neural network has a softmax layer, so that its outputs represent valid probability values. Finally, the failure steps of edges is predicted by picking the index with the highest predicted probability value,

$$\forall e \in E', \quad \hat{f}_e = \arg \max_{t=0, \dots, T-1} \hat{p}_{e,t}. \quad (6)$$

Having defined the model architecture, we explain methods to generate data for training and testing in the next section.

III. DATA SYNTHESIS

We generate cascade data using the cascading failure simulator (CFS) oracle proposed in [10]. Data generated from other oracles or historic data obtained from utility records can also be used on our model without any changes to its architecture.

Algorithm 1 Simulating failure cascade using the CFS oracle.

Input: Grid topology $G = (V, E)$, initial contingency $s[0]$, capacity values $(c_e)_{e \in E}$, and power injection $(P_v)_{v \in V}$.
Output: The branch failure steps in the cascade $(f_e)_{e \in E}$.
initialize $t \leftarrow 0$; overloaded \leftarrow true.
while overloaded **do**
 1) $E \leftarrow \{e \in E : s_e[t] = 1\}$.
 2) Detect the formed islands (disconnected sub-graphs).
 for each island **do**
 a) Specify a bus as the slack bus of the island.
 b) Rebalance the power injection within the island.
 c) Recompute the power flows g_e within the island.
 end for
 3) $s[t+1] \leftarrow s[t]; \quad \forall e \in E : g_e > c_e, s_e[t+1] \leftarrow 0$.
 4) **if** $s[t+1] = s[t]$ **then** overloaded \leftarrow false **end if**
 5) $t \leftarrow t + 1$.
end while
return $f = \sum_t s[t]$.

As summarized in Algorithm 1, the CFS oracle simulates the cascade process for a given initial contingency $s[0]$ and power injection values $(P_v)_{v \in V}$, and outputs the failure steps $f := (f_e)_{e \in E}$. The CFS oracle treats branch failures deterministically, a branch e is treated to be failed whenever the power flow g_e through it crosses its given capacity value c_e . We implement the CFS oracle in MATLAB, where we use the MATPOWER toolbox [31] to get the graph topology, default power injection values, and branch capacity values. We use its DC power flow solver to calculate the branch power flows.

Furthermore, using this CFS oracle, we generate a data pool \mathcal{D} of size M as summarized in Algorithm 2. Each sample in the data pool \mathcal{D} is a tuple $(s[0], (P_v)_{v \in V}, (f_e)_{e \in E})$ containing a random initial contingency, randomly scaled power injection values, and the corresponding cascade failure steps respectively. Specifically, in each sample, we first generate random $|E| - 2$ initial contingencies $s[0]$ by selecting two branches randomly, say e_1, e_2 , and setting their states to failed $s_{e_1}[0] = s_{e_2}[0] = 0$. Then, we scale the default power injection values obtained from the MATPOWER toolbox uniformly across all nodes by a random scaling value $\alpha \sim \text{Unif}[1, 2]$. We then use the CFS oracle described in Algorithm 1 to get the cascade failure steps $(f_e)_{e \in E}$ for each sample.

Algorithm 2 Generating the failure cascade data pool.

Input: Grid topology $G = (V, E)$, default node power injection values $(P_v^0)_{v \in V}$, and required data pool size M .

Output: The failure cascade data pool \mathcal{D} .

initialize $\mathcal{D} \leftarrow \{\}$.

while $|\mathcal{D}| < M$ **do**

- 1) $s[0] \leftarrow \text{Random } |E| - 2 \text{ initial contingency.}$
- 2) $\alpha \leftarrow \text{Unif}[1, 2]; \quad \forall v \in V, P_v \leftarrow \alpha P_v^0.$
- 3) $(f_e)_{e \in E} \leftarrow \text{CFS oracle's output for } (s[0], (P_v)_{v \in V}, G).$
- 4) $\mathcal{D} \leftarrow \mathcal{D} \text{ appended with } (s[0], (P_v)_{v \in V}, (f_e)_{e \in E}).$

end while

return \mathcal{D} .

In the next section, we discuss the details of model training with this data and present their performance results.

IV. RESULTS

We generate two data pools, one for IEEE89 system and another for IEEE118 system, each containing 200,000 cascade sequence samples simulated on random initial contingencies and random uniform scaling values. We get the required graph topology, default power injection values, and branch capacities (we set the unavailable capacities to twice the default power flows through the branches) from the MATPOWER toolbox. In both cases, we split the dataset \mathcal{D} into train \mathcal{D}_{train} (90%) and test \mathcal{D}_{test} (10%) sets. We then train two instances of our proposed model, one for IEEE89 and another for IEEE118 systems¹. We build the instances in Python using the PyTorch library [32]. We train the models with an Adam optimizer and try to minimize the cross entropy loss between predicted and true failure steps averaged on all branches and batch samples.

We use the performance of the influence model [29] as a benchmark since this model can evaluate metrics in almost the same granularity level as the GNN model. Note, however, that the influence model does not take node power injection values into consideration, making it specific to a single loading profile. Hence, it is impossible to generalize a single influence model over all load scaling values. Thus, for a given IEEE system, we build multiple influence model instances, each

trained on a unique load scaling value, and compare our single generalized GNN model against them. The metrics we investigate can be categorized into two types: graph-level and branch-level metrics.

1) *Graph Level Metrics:* Here, we present several metrics, in increasing order of prediction granularity, that capture the graph-level prediction performance of the trained model.

a) *Failure Size Error Rate:* The cascade failure size is defined as the number of branches in the failed state at the end of a cascade. Let $\mathcal{D}_{test}^\alpha \subset \mathcal{D}_{test}$ be the set of test samples whose injection load scaling is α . For a sample $d \in \mathcal{D}_{test}^\alpha$ (with a random $|E| - 2$ initial contingency), if $E_{failed}^d \subset E$ is the set of edges that have truly failed and $\hat{E}_{failed}^d \subset E$ is the set of edges predicted to have failed, then the graph-level failure size error rate l_{size}^α at a load scaling value α is defined as $l_{size}^\alpha = \frac{1}{|\mathcal{D}_{test}^\alpha|} \sum_{d \in \mathcal{D}_{test}^\alpha} ||E_{failed}^d| - |\hat{E}_{failed}^d|| / |E_{failed}^d|$.

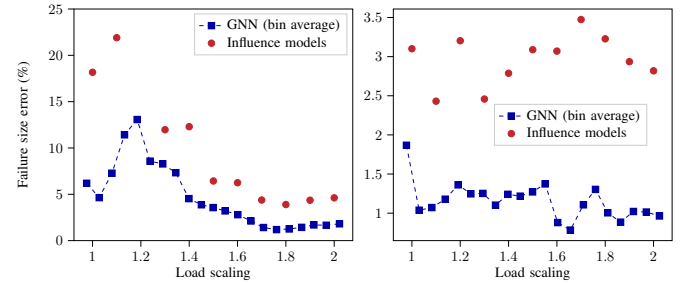


Fig. 2: Failure size error rates l_{size}^α of various models for IEEE89 (left) and IEEE118 (right) against scaling values α .

Fig. 2 shows the failure size error rates of two instances of the GNN model trained and tested on IEEE89 (left) and IEEE118 (right) cases respectively, along with the error rates of multiple influence model instances specific to the corresponding scaling value (recall the influence model's inability to generalize over variable load profiles). As can be seen, even though we are comparing the same generic GNN model to multiple load-specific influence models, the GNN model has lower error rates at all loading values.

b) *Final State Error Rate:* If the true final state of a cascade sample $d \in \mathcal{D}_{test}$ is $s^d[T] = (s_e^d[T])_{e \in E}$, and its predicted final state is $\hat{s}^d[T] = (\hat{s}_e^d[T])_{e \in E}$, then the graph-level final state error rate l_{state}^α at a load scaling value α is defined as $l_{state}^\alpha = \frac{1}{|\mathcal{D}_{test}^\alpha|} \sum_{d \in \mathcal{D}_{test}^\alpha} \frac{1}{|E|} \sum_{e \in E} |s_e^d[T] - \hat{s}_e^d[T]|$.

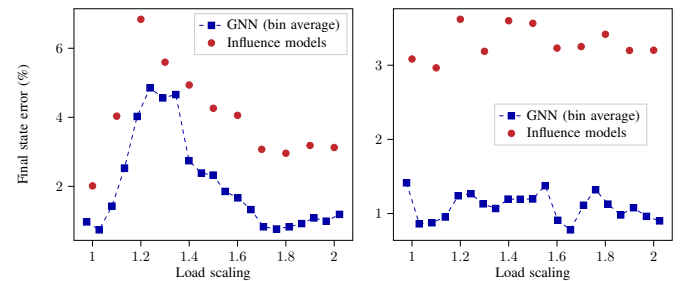


Fig. 3: Final state error rates l_{state}^α of various models for IEEE89 (left) and IEEE118 (right) against scaling values α .

¹See implementations at <https://github.com/sathwikchadaga/failure-cascade>.

Fig. 3 shows the final state error rates of two instances of the GNN model for IEEE89 (left) and IEEE118 (right) systems respectively, along with the error rates of different load-specific influence models. As can be seen, the GNN model is better by around 2% (nearly a factor of 2) than all the load-specific influence models.

c) *Failure Step Error Rate:* In a cascade sample $d \in \mathcal{D}_{test}^\alpha$, if the true failure steps are $f^d = (f_e^d)_{e \in E}$ and the predicted failure steps are $\hat{f}^d = (\hat{f}_e^d)_{e \in E}$, then the graph-level failure step error $l_{failure-step}^\alpha$ at a load scaling α is defined as $l_{failure-step}^\alpha = \frac{1}{|\mathcal{D}_{test}^\alpha|} \sum_{d \in \mathcal{D}_{test}^\alpha} \frac{1}{|E|} \sum_{e \in E} |f_e^d - \hat{f}_e^d|$.

Fig. 4 shows the failure step error rates for two instances of the GNN models for IEEE89 (left) and IEEE118 (right) systems respectively, along with the error rates of different load-specific influence models. For both buses, the generic GNN model outperforms all the load-specific influence models.

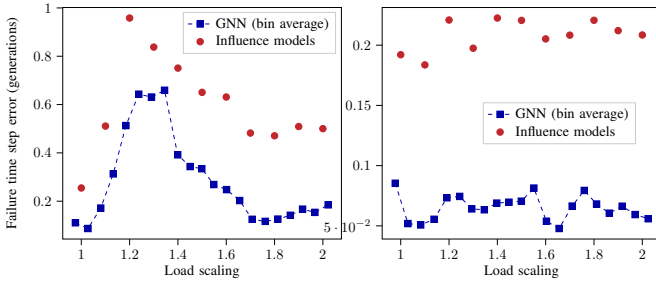


Fig. 4: Failure step error rates $l_{failure-step}^\alpha$ of various models for IEEE89 (left) and IEEE118 (right) against load scaling α .

2) *Branch Level Metrics:* In this subsection, we present several branch metrics that depict the model's accuracy in prediction of branch features. But first, we define the *branch failure frequency* $l_{freq,e}$ for any branch $e \in E$. Let $\mathcal{D}_{failed}^e \subset \mathcal{D}_{train}$ be the set of train samples where the branch has eventually failed in the cascade but not as part of the initial contingency, then we define $l_{freq,e} = |\mathcal{D}_{failed}^e|/|\mathcal{D}_{train}|$. This value captures the prediction difficulty of branches. For example, predicting the features of a branch that rarely fails is easier than a branch that fails half the time.

a) *Branch Final State Error Rate:* Let $\mathcal{D}_{wrong}^e \subset \mathcal{D}_{test}$ be the set of test samples in which the model wrongly predicted the final state of edge $e \in E$. Further, let $\mathcal{D}_{initial}^e \subset \mathcal{D}_{wrong}^e$ be the samples in which branch e failed as part of the initial contingency. We do not count such samples as predicting their states is trivial, hence we define the branch final state error as the ratio $l_{state,e} = |\mathcal{D}_{wrong}^e|/(|\mathcal{D}_{test}| - |\mathcal{D}_{initial}^e|)$.

Fig. 5 shows the final state error rate for two instances of the GNN model trained and tested on IEEE89 (left) and IEEE118 (right) datasets respectively. The error rates are averaged over all test samples containing random initial contingencies and random load scaling values. As can be seen, the branch final state prediction error rate by the GNN model is below 6% and 2.5% at all branches for IEEE89 and IEEE118 systems respectively.

Further, the error plot in Fig. 5 is generated by averaging over random scaling values in $[1, 2]$, which demonstrates that

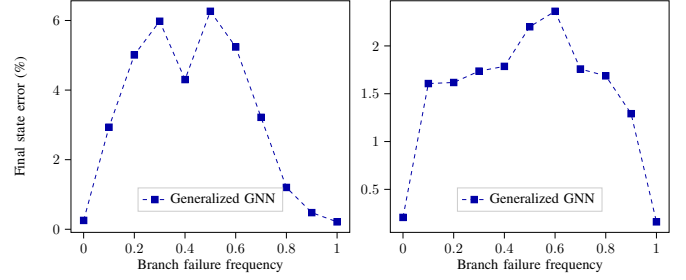


Fig. 5: Branch final state prediction error rate $l_{state,e}$ for IEEE89 (left) and IEEE118 (right) averaged over all load scaling values $[1, 2]$ against branch failure frequencies $l_{freq,e}$.

the GNN model can be generalized over variable load profiles. However, note that the plot does not contain the performance of a reference influence model, again because of the influence model's inability to generalize over variable load profiles. Hence, in order to benchmark the performance, Fig. 6 plots the final state error rates of the same GNN model, tested on three example load scaling values of 1.40, 1.50, and 1.90, against different instances of the influence model built specifically for those load scaling values of 1.40, 1.50, and 1.90.

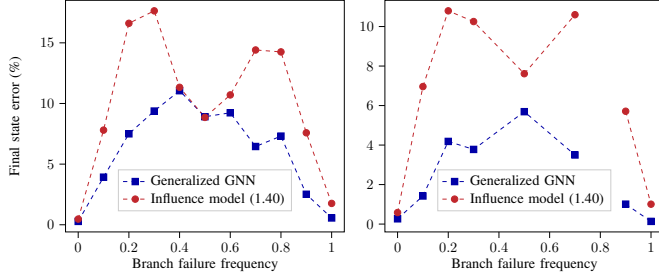
As seen from these plots, the generic GNN model outperforms the load-specific instances of the influence model at almost all branches, beating the influence model by almost 10% in some cases. However, the influence model performs better than the GNN model at some branches with failure frequencies close to 0.5.

b) *Branch Failure Step Error Rate:* Let $\mathcal{D}_{failed}^e \subset \mathcal{D}_{test}$ be the set of test samples where the branch $e \in E$ has eventually failed in the cascade but not as part of the initial contingency. Say, the true failure step of the branch e in sample $d \in \mathcal{D}_{failed}^e$ is f_e^d and the predicted state is \hat{f}_e^d , then we define the branch failure step error rate as $l_{failure-step,e} = \frac{1}{|\mathcal{D}_{failed}^e|} \sum_{d \in \mathcal{D}_{failed}^e} |\hat{f}_e^d - f_e^d|$.

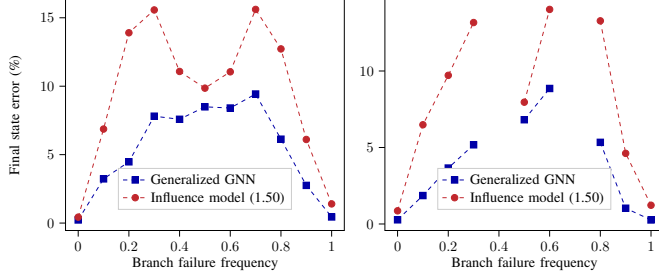
Fig. 7 shows the branch failure step error rate $l_{failure-step,e}$ averaged across random initial contingencies and random load scaling values. As seen in the plot, the failure step error rate is in the order of 0.01 time steps. The significantly low error performance when averaged over random scaling values demonstrates the GNN's generalization capability.

The average plot in Fig. 7 does not contain the performance of a reference influence model because of its inability to generalize over variable load profiles. Hence, to benchmark the performance of our model, Fig. 8 plots the branch failure step error rates of the same GNN model, tested on three example load scaling values of 1.40, 1.50, and 1.90, and compares it to different influence model instances built specifically for those scaling values of 1.40, 1.50, and 1.90.

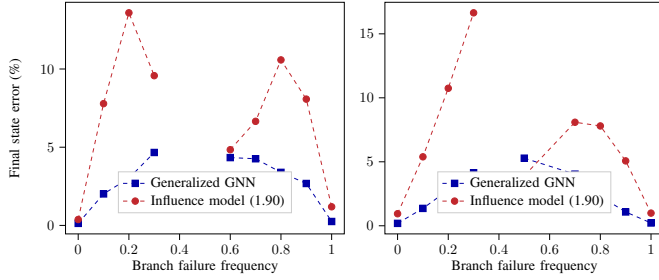
As seen from the plots, the branch failure step prediction performance of the generic GNN model is significantly better than the load-specific influence model instances. In the influence model, when doing state prediction in a step-by-step manner, the errors that occur in initial steps propagate to later steps, thereby accumulating to a large final error. This is completely avoided by the GNN model since it predicts



(a) Load scaling = 1.40 for IEEE89 (left) and IEEE118 (right).



(b) Load scaling = 1.50 for IEEE89 (left) and IEEE118 (right).



(c) Load scaling = 1.90 for IEEE89 (left) and IEEE118 (right).

Fig. 6: Branch final state prediction error rate $l_{state,e}$ for IEEE89 (left) and IEEE118 (right) for various load scaling plotted against branch failure frequencies $l_{freq,e}$.

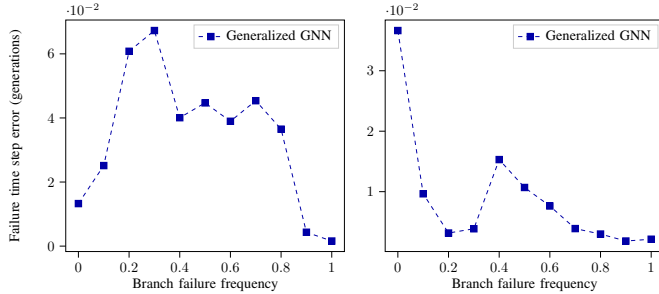
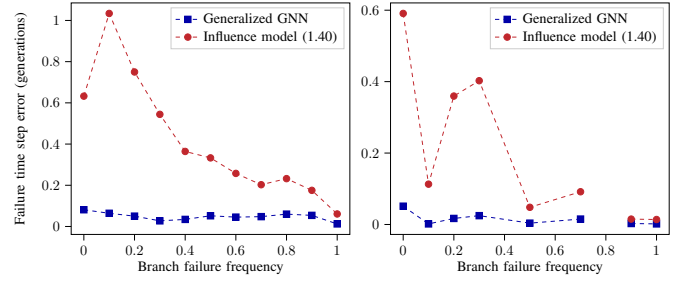
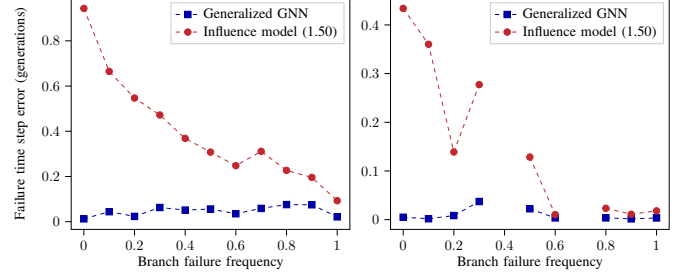


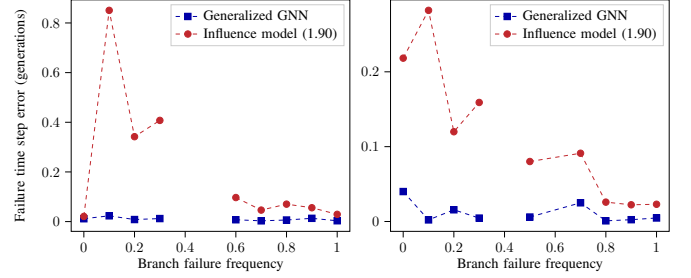
Fig. 7: Branch failure steps prediction error $l_{failure-step,e}$ for IEEE89 (left) and IEEE118 (right) averaged over all scaling [1, 2] against failure frequencies $l_{freq,e}$.



(a) Load scaling = 1.40 for IEEE89 (left) and IEEE118 (right).



(b) Load scaling = 1.50 for IEEE89 (left) and IEEE118 (right).



(c) Load scaling = 1.90 for IEEE89 (left) and IEEE118 (right).

Fig. 8: Branch failure steps prediction error rate $l_{failure-step,e}$ for IEEE89 (left) and IEEE118 (right) for various load scaling plotted against branch failure frequencies $l_{freq,e}$.

the failure steps directly. We believe this causes the GNN to outperform the influence models in this metric.

3) *Runtime Analysis:* We perform a runtime analysis to demonstrate how the GNN model can harvest the power of GPUs to predict cascade sequences much faster than the flow-based simulation methods. We run cascade predictions on 11,000 test samples with the CFS oracle, the influence model, and the GNN model. Table I summarizes the runtime results. The CFS oracle cannot be run on a GPU, hence it was tested in MATLAB 2019a on an Intel(R) Core(TM) i9-7920X CPU@2.90GHz processor with 128GB of installed memory. Further, influence and GNN models were tested on an NVIDIA GeForce RTX 2080 Ti GPU with 11GB of total memory.

TABLE I: Prediction time in seconds per 1000 samples.

	CFS oracle	Influence model	GNN model
IEEE89	24.18	2.35	0.53
IEEE118	62.54	1.86	0.28

It can be seen that the time taken by the influence and GNN models are significantly lower than the CFS oracle. In the influence model, the matrix multiplications can be

sped up using a GPU. However, because of its step-by-step prediction nature, each cascade prediction lasts for a variable number of steps. Hence, we cannot run multiple predictions simultaneously with the influence model unlike the fully parallel GNN model. Hence, the GNN model is almost four times faster than the influence model.

V. CONCLUSION

We considered the problem of predicting the failure cascade sequence due to branch failures given the initial contingency, power injection values, and grid topology. We proposed a flow-free graph neural network model that predicts the grid states at every generation of a cascade, without requiring power flow calculations. We showed that the model, in addition to being generic over randomly scaled loading values, outperforms the influence models that were built specifically for their corresponding loading profiles. Finally, we presented a runtime analysis to show that the model is faster by almost two orders of magnitude than the flow-based cascading failure simulator.

ACKNOWLEDGMENT

This work was supported by NSF grants CNS-1735463 and CNS-2106268, and by a research award from the C3.ai Digital Transformation Institute.

REFERENCES

- [1] J. Kim, "Increasing Power Outages Don't Hit Everyone Equally," 2023. Scientific American. Available: <https://www.scientificamerican.com/article/increasing-power-outages-dont-hit-everyone-equally> (accessed: 8-Aug-2023).
- [2] Climatecentral.org, "Surging Weather-related Power Outages," 2022. Online. Available: <https://www.climatecentral.org/climate-matters/surging-weather-related-power-outages> (accessed: 8-Aug-2023).
- [3] C. Clifford, "Why america's outdated energy grid is a climate problem," 2023. CNBC. Available: <https://www.cnbc.com/2023/02/17/why-americas-outdated-energy-grid-is-a-climate-problem.html> (accessed: 8-Aug-2023).
- [4] K. H. LaCommare, J. H. Eto, L. N. Dunn, and M. D. Sohn, "Improving the estimated cost of sustained power interruptions to electricity customers," *Energy*, vol. 153. Elsevier BV, pp. 1038–1047, Jun. 2018. doi: 10.1016/j.energy.2018.04.082.
- [5] B. Stone Jr. et al., "How Blackouts during Heat Waves Amplify Mortality and Morbidity Risk," *Environmental Science & Technology*, vol. 57, no. 22. American Chemical Society (ACS), pp. 8245–8255, May 23, 2023. doi: 10.1021/acs.est.2c09588.
- [6] H. Ren and I. Dobson, "Using Transmission Line Outage Data to Estimate Cascading Failure Propagation in an Electric Power System," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 9, pp. 927–931, Sept. 2008, doi: 10.1109/TCSII.2008.924365.
- [7] P. D. H. Hines, I. Dobson and P. Rezaei, "Cascading Power Outages Propagate Locally in an Influence Graph That is Not the Actual Grid Topology," in *IEEE Transactions on Power Systems*, vol. 32, no. 2, pp. 958–967, March 2017, doi: 10.1109/TPWRS.2016.2578259.
- [8] S. Soltan, D. Mazauric and G. Zussman, "Analysis of Failures in Power Grids," in *IEEE Transactions on Control of Network Systems*, vol. 4, no. 2, pp. 288–300, June 2017, doi: 10.1109/TCNS.2015.2498464.
- [9] H. Cetinay, S. Soltan, F. A. Kuipers, G. Zussman and P. Van Mieghem, "Comparing the Effects of Failures in Power Grids Under the AC and DC Power Flow Models," in *IEEE Transactions on Network Science and Engineering*, vol. 5, no. 4, pp. 301–312, 1 Oct.–Dec. 2018, doi: 10.1109/TNSE.2017.2763746.
- [10] M. J. Eppstein and P. D. H. Hines, "A "Random Chemistry" Algorithm for Identifying Collections of Multiple Contingencies That Initiate Cascading Failure," in *IEEE Transactions on Power Systems*, vol. 27, no. 3, pp. 1698–1705, Aug. 2012, doi: 10.1109/TPWRS.2012.2183624.
- [11] J. Qi, J. Wang and K. Sun, "Efficient Estimation of Component Interactions for Cascading Failure Analysis by EM Algorithm," in *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3153–3161, May 2018, doi: 10.1109/TPWRS.2017.2764041.
- [12] J. Xie, I. Alvarez-Fernandez and W. Sun, "A Review of Machine Learning Applications in Power System Resilience," 2020 IEEE Power & Energy Society General Meeting (PESGM), Montreal, QC, Canada, 2020, pp. 1–5, doi: 10.1109/PESGM41954.2020.9282137.
- [13] Y. Yang, Z. Yang, J. Yu, B. Zhang, Y. Zhang, and H. Yu, "Fast Calculation of Probabilistic Power Flow: A Model-Based Deep Learning Approach," *IEEE Transactions on Smart Grid*, vol. 11, no. 3. Institute of Electrical and Electronics Engineers (IEEE), pp. 2235–2244, May 2020. doi: 10.1109/tsg.2019.2950115.
- [14] Y. Du, F. Li, J. Li and T. Zheng, "Achieving 100x Acceleration for N-1 Contingency Screening With Uncertain Scenarios Using Deep Convolutional Neural Network," in *IEEE Transactions on Power Systems*, vol. 34, no. 4, pp. 3303–3305, July 2019, doi: 10.1109/TPWRS.2019.2914860.
- [15] S. Gupta, R. Kambli, S. Wagh and F. Kazi, "Support-Vector-Machine-Based Proactive Cascade Prediction in Smart Grid Using Probabilistic Framework," in *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2478–2486, April 2015, doi: 10.1109/TIE.2014.2361493.
- [16] R. A. Shuvro, P. Das, M. M. Hayat and M. Talukder, "Predicting Cascading Failures in Power Grids using Machine Learning Algorithms," 2019 North American Power Symposium (NAPS), Wichita, KS, USA, 2019, pp. 1–6, doi: 10.1109/NAPS46351.2019.9000379.
- [17] H. Zhang, T. Ding, J. Qi, W. Wei, J. P. S. Catalão and M. Shahidehpour, "Model and Data Driven Machine Learning Approach for Analyzing the Vulnerability to Cascading Outages With Random Initial States in Power Systems," in *IEEE Transactions on Automation Science and Engineering*, 2022, doi: 10.1109/TASE.2022.3204273.
- [18] R. Pi, Y. Cai, Y. Li and Y. Cao, "Machine Learning Based on Bayes Networks to Predict the Cascading Failure Propagation," in *IEEE Access*, vol. 6, pp. 44815–44823, 2018, doi: 10.1109/ACCESS.2018.2858838.
- [19] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv*, 2016. doi: 10.48550/ARXIV.1609.02907.
- [20] B. Donon, B. Donnot, I. Guyon, and A. Marot, "Graph Neural Solver for Power Systems," 2019 International Joint Conference on Neural Networks (IJCNN). IEEE, Jul. 2019. doi: 10.1109/ijcnn.2019.8851855.
- [21] D. Wang, K. Zheng, Q. Chen, G. Luo, and X. Zhang, "Probabilistic Power Flow Solution with Graph Convolutional Network," 2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe). IEEE, Oct. 26, 2020. doi: 10.1109/isgt-europe47291.2020.9248786.
- [22] J. B. Hansen, S. N. Anfinsen, and F. M. Bianchi, "Power Flow Balancing with Decentralized Graph Neural Networks," *arXiv*, 2021, doi: 10.48550/ARXIV.2111.02169.
- [23] B. Donon, R. Clément, B. Donnot, A. Marot, I. Guyon, and M. Schoenauer, "Neural networks for power flow: Graph neural solver," *Electric Power Systems Research*, vol. 189. Elsevier BV, p. 106547, Dec. 2020. doi: 10.1016/j.epsr.2020.106547.
- [24] A. B. Jeddi and A. Shafieezadeh, "A Physics-Informed Graph Attention-based Approach for Power Flow Analysis," 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, Dec. 2021. doi: 10.1109/icmla52953.2021.00261.
- [25] C. Kim, K. Kim, P. Balaprakash, and M. Anitescu, "Graph Convolutional Neural Networks for Optimal Load Shedding under Line Contingency," 2019 IEEE Power & Energy Society General Meeting (PESGM). IEEE, Aug. 2019. doi: 10.1109/pesgm40551.2019.8973468.
- [26] Y. Zhu, Y. Zhou, W. Wei, and L. Zhang, "Real-Time Cascading Failure Risk Evaluation With High Penetration of Renewable Energy Based on a Graph Convolutional Network," *IEEE Transactions on Power Systems*. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–12, 2022. doi: 10.1109/tpwrs.2022.3213800.
- [27] B. Jhun, H. Choi, Y. Lee, J. Lee, C. H. Kim, and B. Kahng, "Prediction and mitigation of nonlocal cascading failures using graph neural networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 33, no. 1. AIP Publishing, p. 013115, Jan. 2023. doi: 10.1063/5.0107420.
- [28] A. Dwivedi and A. Tajer, "GRNN-Based Real-Time Fault Chain Prediction," *IEEE Transactions on Power Systems*. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–13, 2023. doi: 10.1109/tpwrs.2023.3258740.
- [29] X. Wu, D. Wu and E. Modiano, "Predicting Failure Cascades in Large Scale Power Systems via the Influence Model Framework," in *IEEE*

Transactions on Power Systems, vol. 36, no. 5, pp. 4778-4790, Sept. 2021, doi: 10.1109/TPWRS.2021.3068409.

- [30] Y. Zhu, Y. Zhou, W. Wei, and N. Wang, "Cascading Failure Analysis Based on a Physics-Informed Graph Neural Network," IEEE Transactions on Power Systems. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–10, 2022. doi: 10.1109/tpwrs.2022.3205043.
- [31] R. D. Zimmerman, C. E. Murillo-Sanchez, and R. J. Thomas, "Matpower: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education," Power Systems, IEEE Transactions on, vol. 26, no. 1, pp. 12–19, Feb. 2011. doi: 10.1109/TPWRS.2010.2051168.
- [32] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library." arXiv, 2019. doi: 10.48550/ARXIV.1912.01703.