# LEARNING TO BOOST NETWORK RESILIENCE WITHOUT RICH FEATURES VIA INDUCTIVE TOPOLOGY-INSPIRED GRAPH NEURAL NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Improving the resilience of a complex network is a fundamental problem in network science, typically achieved via successive degree-preserving edge rewiring operations. Existing resilience optimization methods share the limitation of being transductive: a learned edge rewiring strategy from one graph cannot be generalized to another. Such a limitation cannot be trivially addressed by existing learning-based approaches that require rich initial features for graph neural networks (GNNs) to learn useful representations. We found traditional GNNs degenerate remarkably on the resilience task with only pure topologies available. Therefore, a natural question is how to learn useful representations from purely topological graphs to achieve inductivity, such that the learned edge rewiring strategies can be generalized to optimize new graphs without searching from scratch. In this work, inspired by persistent homology, we specifically design a variant of GNN called FireGNN for the resilience task that learns meaningful representations from pure graph topologies. Equipped with FireGNN, we develop **ResiNet**, the first end-to-end inductive network resilience optimization method to discover **Resi**lient **Net**work topologies while balancing network utility. ResiNet reformulates resilience optimization as a Markov decision process equipped with edge rewiring action space and learns to select correct edges for edge rewiring successively. Extensive experiments demonstrate that ResiNet achieves a near-optimal resilience gain on various graphs while balancing the utility and outperforms existing approaches by a large margin.

## 1 INTRODUCTION

*Network resilience*, in the context of network science, is a measurement characterizing the ability of a system to defend itself from such failures and attacks (Schneider et al., 2011). Network systems, such as infrastructure systems and supply chains, are vulnerable to malicious attacks. Therefore, studying the resilience of complex networks has found wide applications in many fields, such as ecology (Sole & Montoya, 2001), biology (Motter et al., 2008), economics (Haldane & May, 2011) and engineering (Albert et al., 2004). A resilient network should continue to function and maintain an acceptable level of utility when the network partially fails.

Current network resilience optimization methods improve the resilience of networks over graph topologies based on an atomic operation called *edge rewiring* (Schneider et al., 2011; Chan & Akoglu, 2016; Rong & Liu, 2018). Concretely, for a given graph $G = (V, E)$ and two existing edges $AC$ and $BD$, an edge rewiring operation[1] alters the graph structure by removing $AC$ and $BD$ and adding $AB$ and $CD$, where $AC, BD \in E$ and $AB, CD, AD, BC \notin E$. Edge rewiring operation has some nice properties against simply addition or deletion of edges: 1) it preserves the node degree and the total degree of a graph, while addition/deletion may cause undesired change; 2) it achieves minimal changes in terms of utility measure in terms of graph Laplacian, while addition/deletion may lead to an unpredictable network utility degradation (Jaume et al., 2020; Ma et al., 2021).

Traditional non-learning-based resilience optimization methods typically fall into the categories of evolutionary computation (Zhou & Liu, 2014) or heuristic-based (Schneider et al., 2011; Chan & Akoglu, 2016; Rong & Liu, 2018; Yazıcıoğlu et al., 2015), with the following limitations:

[1] A few works have a different definition for the edge rewiring operation.

- *Transductivity*. Traditional methods are transductive since they search the resilience topology on a particular problem instance. This search procedure is performed for every individual graph without generalization.
- *Local optimality*. It is NP-hard to combinatorially choose the edges to rewire to obtain the globally optimal resilience (Mosk-Aoyama, 2008). Previous studies predominantly adopt greedy-like algorithms, yielding local optimality in practice (Chan & Akoglu, 2016).
- *Utility Loss*. Rewiring operation in resilience optimization may potentially lead to an degradation of the network utility, which may jeopardize the network functioning.

Although the learning-based paradigm equipped with GNNs has proved powerful for a large variety of graph tasks with rich features (Li et al., 2018; Joshi et al., 2019; Fu et al., 2020; Khalil et al., 2017; Nazari et al., 2018; Peng et al., 2020; Yu et al., 2019), it still remains opaque how to adapt such approaches to resilience optimization with only topological structures available. One may infer that such a lack of feature can significantly hinder the learning ability, and a case agreeing with this has been discovered in solving the traveling salesman problem (TSP): Boffa et al. (2022) showed that the performance of GNNs degenerates largely when node coordinate features are missing (but pairwise distance is given), compared to the case when coordinates are available. Although Boffa et al. (2022) adopted a distance encoding strategy (Li et al., 2020b) to alleviate the performance gap in TSP, we empirically found that this encoding strategy does not work on the more challenging resilience task (See Sec. 5.2). Therefore, it is demanding to devise a novel GNN that can be applicable for the resilience task without rich features. Readers are referred to Appendix E for a more detailed analysis.

In this work, we present ResiNet, the first *inductive* learning-based method for discovering resilient network topology using successive edge rewiring operations. To overcome the above limitation of GNNs in modeling graphs without rich features, we specially design a topology-oriented variant of GNN called **Fi**l**tr**ation **e**nhanced **GNN** (FireGNN). FireGNN creates a series of subgraphs (the filtration) by successively removing the node with the highest degree from the graph and then learns to aggregate the node representations from each subgraph. This filtration process innovation is inspired by persistent homology and the approximation of the persistence diagram (Edelsbrunner & Harer, 2008; Aktas et al., 2019; Hofer et al., 2020).

The main contributions of this paper are summarized as follows:

1) We propose ResiNet, the first learning-based model to boost network resilience inductively in a degree-preserving manner with moderate loss of the network utility. ResiNet forms resilience optimization into a successive sequential decision process of edge rewiring operations. Extensive experiments show that ResiNet achieves near-optimal resilience gain while balancing network utilities. Existing approaches are outperformed by a large margin.
2) FireGNN, our technical innovation serving as the graph feature extractor, can learn meaningful representations from pure topological structures, which provides sufficient training signals to learn an RL agent to perform successive edge rewiring operations inductively.

## 2 Related work

**Network resilience.** Modern network systems are threatened by various malicious attacks, such as the destruction of critical nodes, critical connections and critical subset of the network via heuristics/learning-based attack (Fan et al., 2020; Zhao et al., 2021; Zhang et al., 2017; Holme et al., 2002; Iyer et al., 2013; Grassia et al., 2021; Fan et al., 2020; Medya et al., 2020). Network resilience was proposed and proved as a suitable measurement for describing the robustness and stability of a network system under such attacks (Schneider et al., 2011). Around optimizing network resilience, various defense strategies have been proposed to protect the network functionality from crashing and preserve network's topologies to some extent. Commonly used manipulations of defense include adding additional edges (Li et al., 2019; Carchiolo et al., 2019), protecting vulnerable edges (Wang et al., 2014) and rewiring two edges (Schneider et al., 2011; Chan & Akoglu, 2016; Buesser et al., 2011). Among these manipulations, edge rewiring fits well to real-world applications as it induces fewer functionality changes to the original network and does not impose additional loads to the vertices (degree-preserving) (Schneider et al., 2011; Rong & Liu, 2018; Yazıcıoğlu et al., 2015). By now, there has been no learning-based inductive strategy for the resilience task.

**GNNs for graph-related tasks.** GNNs are powerful tools to learn from relational data, providing meaningful representations for the downstream task. Several successful applications using GNNs as backbones include node classification (Kipf & Welling, 2017; Hamilton et al., 2017; Velickovic et al., 2018), link prediction (Li et al., 2020a; Kipf & Welling, 2017; Hamilton et al., 2017), graph property estimation (Xu et al., 2019; Kipf & Welling, 2017; Li et al., 2020a; Bodnar et al., 2021), and combinatorial problems on graphs (e.g., TSP (Li et al., 2018; Joshi et al., 2019; Fu et al., 2020; Khalil et al., 2017; Hudson et al., 2022), vehicle routing problem (Nazari et al., 2018; Peng et al., 2020; Yu et al., 2019), graph matching (Yu et al., 2021) and adversarial attack on GNNs (Ma et al., 2021; Dai et al., 2018)). Although standard message passing GNNs are powerful, their expressive power is upper-bounded by the 1-Weisfeiler-Lehman (1-WL) test (Xu et al., 2019). Many advanced techniques are designed for enabling GNNs with greater expressive power over the 1-WL test, such as distance encoding (Li et al., 2020b) and high-order GNNs (Morris et al., 2019). Such strategies are empirically observed to be more expressive than 1-WL versions under rare feature cases (You et al., 2021; Boffa et al., 2022). For example, distance encoding was employed by Boffa *et al.* in a learning-based TSP solver, achieving better performance than standard GNNs when only pairwise distance is given (Boffa et al., 2022). However, it still obviously under-performs compared to the case when GNNs are fully trained with node coordinates. Till now, it remains unclear to adapt GNNs to graph tasks without rich feature, or even with pure topology, as in the resilience task.

**Extended related work.** The related work on *network resilience and utility*, *multi-views graph augmentation for GNNs*, *deep graph generation*, *graph rewiring* and *graph structure learning* is deferred to Appendix A.

## 3 Problem Definition

An undirected graph is defined as $G = (V, E)$, where $V = \{1, 2, \ldots, N\}$ is the set of $N$ nodes, $E$ is the set of $M$ edges, $A \in \{0, 1\}^{N \times N}$ is the adjacency matrix, and $F \in \mathbb{R}^{N \times d}$ is the $d$-dimensional node feature matrix[2]. The degree of a node is defined as $d_i = \sum_{j=1}^{N} A_{ij}$, and a node with degree 0 is called an isolated node.

Given the network resilience metric $\mathcal{R}(G)$ and the utility metric $\mathcal{U}(G)$ (defined in Sec. 5.1), let $\mathbb{G}_G$ denote the set of graphs with the *same* node degrees as $G$. The objective of boosting the resilience of $G$ is to find a target graph $G^\star \in \mathbb{G}_G$, which maximizes the network resilience while balancing the network utility. Formally, the problem of maximizing network resilience is formulated as

$$G^\star = \underset{G' \in \mathbb{G}_G}{\arg\max}\ \alpha \cdot \mathcal{R}(G^{'}) + (1 - \alpha) \cdot \mathcal{U}(G^{'}),$$

To satisfy the constraint of preserving degree, the edge rewiring operation is the default atomic action for obtaining new graph candidates $G^{'}$ from $G$. At each step, two existing edges $AC$ and $BD$ are first selected. Then the edge rewiring alters the graph structure by removing $AC$ and $BD$ and adding $AB$ and $CD$, where $AB, CD, AD, BC \notin E$. Combinatorially, a total of $T$ successive steps of edge rewiring has the complexity of $O(E^{2T})$.

## 4 Proposed Approach: ResiNet

In this section, we formulate the task of boosting network resilience as a reinforcement learning task by learning to select two edges and rewire them successively. We first present the graph resilience-aware environment design and describe our innovation FireGNN in detail. Finally, we present the graph policy network that guides the edge selection and rewiring process.

### 4.1 Boosting network resilience via edge rewiring as Markov decision process

We formulate the network resilience optimization problem via successive edge rewiring operations into the Markov decision process (MDP) framework. The Markov property denotes that the graph obtained at time step $t + 1$ relies only on the graph at time step $t$ and the rewiring operation, reducing

[2]A graph without rich feature only has the topology structure and the node feature is not available.

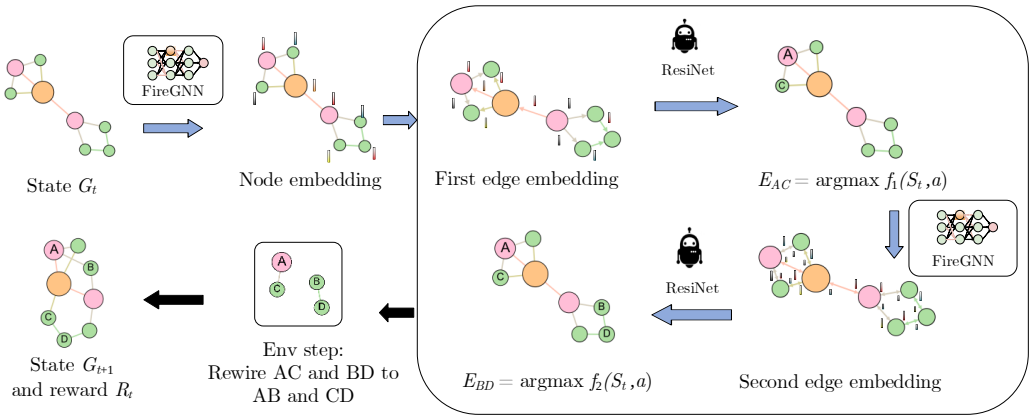
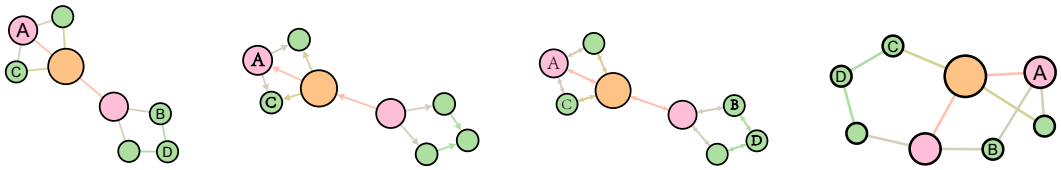



Figure 1: Overview of the architecture of ResiNet to select two edges for rewiring.



(a) Example graph (b) First edge selection (c) Second edge selection (d) Graph after the edge rewiring

Figure 2: Illustration of the edge rewiring operation with the removal of $AC$, $BD$ and the addition of $AB$, $CD$.

the complexity from original $O(E^{2T})$ to $O(TE^2)$. Then we further reduce the complexity to $O(TE)$ by designing an autoregressive edge selection module shown as follows.

As illustrated in Figure 1, the environment performs the resilience optimization in an auto-regressive step-wise way through a sequence of edge rewiring actions guided by ResiNet. Given an input graph, the agent first decides whether to terminate or not. If it chooses not to terminate, it selects one edge from the graph to remove, receives the very edge it just selected as the auto-regression signal, and then selects another edge to remove. Four nodes of these two removed edges are re-combined, forming two new edges to be added to the graph. The optimization process repeats until the agent decides to terminate. The details of the design of the state, the action, the transition dynamics, and the reward are presented as follows.

**State.** The fully observable state is formulated as $S_t = G_t$, where $G_t$ is the current input graph at step $t$. The detailed node feature initialization strategy is given in Appendix C.3.

**Action.** ResiNet is equipped with a node permutation-invariant, variable-dimensional action space. Given a graph $G_t$, the action $a_t$ is to select two edges and the rewiring order. As is shown in Figure 2, the agent first chooses an edge $e_1 = AC$ and a direction $A \to C$. Then conditioning on the state, $e_1$, and the direction the agents chooses an edge $e_2 = BD$ such that $AB, CD, AD, BC \notin E$ and a direction $B \to D$. The heads of the two edges reconnect as a new edge $AB$, and so does the tail $CD$. Although $G_t$ is undirected, we propose to consider the artificial edge directions, which effectively avoids the redundancy in representing action space since $A \to C$, $B \to D$ and $C \to A$, $D \to B$ refer to the same rewiring operation. Therefore, our proposed action space reduces the size of the original action space by half and still leads to complete action space. In this way, the action space is the set of all feasible pairs of $(e_1, e_2) \in E^2$, with a variable size no larger than $2|E|(|E| - 1)$.

**Transition dynamics.** The formulation of the action space implies that if the agent does not terminate at step $t$, the selected action must form an edge rewiring. This edge rewiring is executed by the environment and the graph transits to the new graph.

Note that in some other work, infeasible operations are also included in the action space (to make the action space constant through the process) (You et al., 2018; Trivedi et al., 2020). This reduces the sample efficiency and causes biased gradient estimations (Huang & Ontañón, 2020). ResiNet takes advantage of the state-dependent variable action space composed of only feasible operations.

**Reward.** ResiNet aims to optimize the resilience while balancing the utility, forming a complicated and possibly unknown objective function. Despite this, by (Wakuta, 1995), an MDP that maximizes a complicated objective is up to an MDP that maximizes the linear combination of resilience and utility for some coefficient factor. This fact motivates us to design the reward as the step-wise gain of such a linear combination as

$$R_t = \alpha \cdot \mathcal{R}(G_{t+1}) + (1-\alpha) \cdot \mathcal{U}(G_{t+1}) - (\alpha \cdot \mathcal{R}(G_t) + (1-\alpha) \cdot \mathcal{U}(G_t)),$$

where $\mathcal{R}(G)$ and $\mathcal{U}(G)$ are the resilience and the utility functions, respectively. The cumulative reward $\sum_{t=0}^{T-1} R_t$ up to time $T$ is then the total gain of such a linear combination.

### 4.2 FIREGNN

Motivated by graph filtration in persistent homology (Edelsbrunner & Harer, 2008), we design the **fil**tr**ated** graph **e**nhanced **GNN** termed FireGNN to model graphs without rich features, or even with only topology. As shown in Figure 3, for a given input graph $G$, FireGNN transforms $G$ from the static version to a temporal version consisting of a sequence of subgraphs, by repeatedly removing the node with the highest degree[3]. Observing a sequence of nested subgraphs of $G$ grants FirGNN the capability to observe how $G$ evolves towards being empty. Then FireGNN aligns and aggregates the node, edge and graph embedding from each subgraph, leading to meaningful representations in node, edge, and graph levels. Formally, the filtration in FireGNN is constructed as

$$G^{(k-1)} = G^{(k)} - v_k, \quad v_k = \operatorname*{argmax}_{v_i \in G^{(k)}} \text{DEGREE}(v_i)$$

$$(V, \emptyset) = G^{(0)} \subset G^{(1)} \subset \cdots \subset G^{(N)} = G$$

$$\tilde{G} = [G^{(0)}, G^{(1)}, \ldots, G^{(N)}],$$

where $G^{(k)}$ denotes the remaining graph after removing $N-k$ nodes with highest node degrees, $v_k$ denotes the node with highest degree in current subgraph $G^{(k)}$, DEGREE$(\cdot)$ measures the node degree, $G^{(N)}$ is the original graph, and $G^{(0)}$ contains no edge. The sequence of the nested subgraphs of $G$ is termed the filtrated graph $\tilde{G}$.

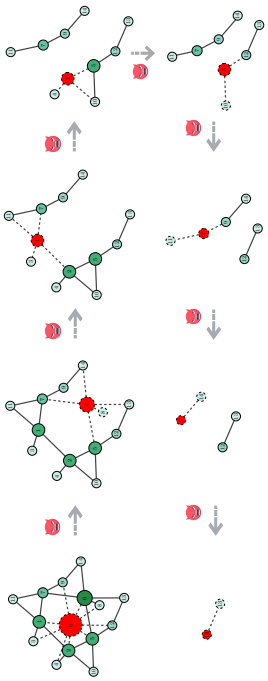


Figure 3: Filtration process in FireGNN.

**Node embedding.** Regular GNN only operates on the original graph $G$ to obtain the node embedding for each node $v_i$ as $h(v_i) = \phi(G^{(N)} = G)_i$, where $\phi(\cdot)$ denotes a standard GNN model. In FireGNN, by using the top $K+1$ subgraphs in a graph filtration, the final node embedding $h(v_i)$ of $v_i$ is obtained by

$$h(v_i) = \text{AGG}_N\left(h^{(N-K)}(v_i), \ldots, h^{(N-1)}(v_i), h^{(N)}(v_i)\right),$$

where $\text{AGG}_N(\cdot)$ denotes a node-level aggregation function, $h^{(k)}(v_i)$ is the node embedding of $i$ in the $k$-th subgraph $G^{(k)}$, and $K \in [N]$. In practice, $h^{(k)}(v_i)$ is discarded when calculating $h(v_i)$ if $v_i$ is isolated or not included in $G^{(k)}$.

**Edge embedding.** The directed edge embedding $h^{(k)}(e_{ij})$ of the edge from node $i$ to node $j$ in each subgraph is obtained by combining the embeddings of the two end vertices in $G^{(k)}$ as

$$h^{(k)}(e_{ij}) = m_f\left(\text{AGG}_{N\to E}\left(h^{(k)}(v_i), h^{(k)}(v_j)\right)\right),$$

[3]Removing the node with the highest degree leads to an efficient minimal number of resultant subgraphs, comparing to the uniformly random removal of nodes.

where $\text{AGG}_{N\to E}(\cdot)$ denotes an aggregation function for obtaining edge embedding from two end vertices (typically chosen from `min`, `max`, `sum`, `difference`, and `multiplication`). $m_f(\cdot)$ is a multilayer perceptron (MLP) model that ensures the consistence between the dimensions of edge embedding and graph embedding.

The final embedding of the directed edge $e_{ij}$ of the filtrated graph $\tilde{G}$ is given by

$$h(e_{ij}) = \text{AGG}_E\left(h^{(N-K)}(e_{ij}), \ldots, h^{(N-1)}(e_{ij}), h^{(N)}(e_{ij})\right) ,$$

where $\text{AGG}_E(\cdot)$ denotes an edge-level aggregation function.

**Graph embedding.** With the node embedding $h^{(k)}(v_i)$ of each subgraph $G^{(k)}$ available, the graph embedding $h^{(k)}(G)$ of each subgraph $G^{(k)}$ is calculated by a readout functions (e.g., `mean`, `sum`) on all non-isolated nodes in $G^{(k)}$ as

$$h^{(k)}(G) = \text{ READOUT}\left(h^{(k)}(v_i)\right) \forall v_i \in G^{(k)} \text{ and } d_i^{(k)} \geq 0 .$$

The final graph embedding of the filtrated graph $\tilde{G}$ is given by

$$h(G) = \text{AGG}_G\left(h^{(N-K)}(G), \ldots, h^{(N-1)}(G), h^{(N)}(G)\right) ,$$

where $\text{AGG}_G(\cdot)$ denotes a graph-level aggregation function.

### 4.3 Edge rewiring policy network

Having presented the details of the graph resilience environment and FireGNN, in this section, we describe the policy network architecture of ResiNet in detail, which learns to select two existing edges for rewiring at each step. At time step $t$, the policy network uses FireGNN as the graph extractor to obtain the directed edge embedding $h(e_{ij}) \in \mathbb{R}^{2|E|\times d}$ and the graph embedding $h(G) \in \mathbb{R}^d$ from the filtrated graph $\tilde{G}_t$, and outputs an action $a_t$ representing two selected rewired edges, leading to the new state $G_{t+1}$ and the reward $R_t$.

To be inductive, we adapt a special autoregressive node permutation-invariant dimension-variable action space to model the selection of edges from graphs with arbitrary sizes and permutations. The detailed mechanism of obtaining the action $a_t$ based on edge embedding and graph embedding is presented as follows, further reducing the complexity from $O(TE^2)$ to $O(TE)$.

**Auto-regressive latent edge selection.** An edge rewiring action $a_t$ at time step $t$ involves the prediction of the termination probability $a_t^{(0)}$ and the selection of two edges ($a_t^{(1)}$ and $a_t^{(2)}$) and the rewiring order. The action space of $a_t^{(0)}$ is binary, however, the selection of two edges imposes a huge action space in $O(|E|^2)$, which is too expensive to sample from even for a small graph. Instead of selecting two edges simultaneously, we decompose the joint action $a_t$ into $a_t = (a_t^{(0)}, a_t^{(1)}, a_t^{(2)})$, where $a_t^{(1)}$ and $a_t^{(2)}$ are two existing edges which do not share any common node (recall that $a_t^{(1)}$ and $a_t^{(2)}$ are directed edges for an undirected graph). Thus the probability of $a_t$ is formulated as

$$\mathbb{P}(a_t|s_t) = \mathbb{P}(a_t^{(0)}|s_t)\mathbb{P}(a_t^{(1)}|s_t, a_t^{(0)})\mathbb{P}(a_t^{(2)}|s_t, a_t^{(0)}, a_t^{(1)}) .$$

**Predicting the termination probability.** The first policy network $\pi_0(\cdot)$ takes the graph embedding as input and outputs the probability distribution of the first action that decides to terminate or not as

$$\mathbb{P}(a_t^{(0)}|s_t) = \pi_0(h(G)) ,$$

where $\pi_0(\cdot)$ is implemented by a two layer MLP. Then $a_t^{(0)} \sim \text{Bernoulli}(\mathbb{P}(a_t^{(0)}|s_t)) \in \{0, 1\}$.

**Selecting edges.** If the signal $a_t^{(0)}$ given by the agent decides to continue to rewire, two edges are then selected in an auto-regressive way. The signal of continuing to rewire $a_t^{(0)}$ is input to the selection of two edges as a one-hot encoding vector $l_c$. The second policy network $\pi_1(\cdot)$ takes the

graph embedding and $l_c$ as input and outputs a latent vector $l_1 \in \mathbb{R}^d$. The pointer network (Vinyals et al., 2015) is used to measure the proximity between $l_1$ and each edge embedding $h(e_{ij})$ in $G$ to obtain the first edge selection probability distribution. Then, to select the second edge, the graph embedding $h(G)$ and the first selected edge embedding $h(e_t^{(1)})$ and $l_c$ are concatenated and fed into the third policy network $\pi_2(\cdot)$. $\pi_2(\cdot)$ obtains the latent vector $l_2$ for selecting the second edge using a respective pointer network. The overall process can be formulated as

$$
\begin{aligned}
l_1 &= \pi_1([h(G), l_c]) \\
\mathbb{P}(a_t^{(1)}|s_t, a_t^{(0)}) &= f_1(l_1, h(e_{ij})),\ \forall e_{ij} \in E \\
l_2 &= \pi_2([h(G), h(e_t^{(1)}), l_c]) \\
\mathbb{P}(a_t^{(2)}|s_t, a_t^{(1)}, a_t^{(0)}) &= f_2(l_2, h(e_{ij})),\ \forall e_{ij} \in E\,,
\end{aligned}
$$

where $\pi_i(\cdot)$ is a two-layer MLP model, $[\cdot,\cdot]$ denotes the concatenation operator, $h(e_t^{(1)})$ is the embedding of the first selected edge at step $t$, and $f_i(\cdot)$ is a pointer network.

# 5 EXPERIMENTS

In this section, we demonstrate the advantages of ResiNet over existing non-learning-based and learning-based methods in achieving superior network resilience, inductively generalizing to unseen graphs, and accommodating multiple resilience and utility metrics. Moreover, we show that FireGNN can learn meaningful representations from graph data without rich features, while current GNNs fail. *Our implementation is already open sourced*.

## 5.1 EXPERIMENTAL SETTINGS

**Datasets.** Synthetic datasets, real EU power network (Zhou & Bialek, 2005) and Internet peer-to-peer networks (Leskovec et al., 2007; Ripeanu et al., 2002) are used to demonstrate the performance of ResiNet in transductive and inductive settings. The details of data generation and the statistics of the datasets are presented in Appendix C.1. Following the conventional experimental settings in network science, the maximal node size is set to be around 1000 (Schneider et al., 2011), taking into account: 1) the high complexity of selecting two edges at each step is $O(E^2)$; 2) evaluating the resilience metric is time-consuming for large graphs.

**Baselines.** We compare ResiNet with existing graph resilience optimization algorithms, including non-learning-based methods and learning-based algorithms. Non-learning-based methods (upper half of Table 1) include the hill climbing (HC) (Schneider et al., 2011), the greedy algorithm (Chan & Akoglu, 2016), the simulated annealing (SA) (Buesser et al., 2011), and the evolutionary algorithm (EA) (Zhou & Liu, 2014). Since there is no previous learning-based baseline, we specifically devise two counterparts based on our method by replacing FireGNN with existing well-known powerful GNNs (DE-GNN (Li et al., 2020b), $k$-GNN (Morris et al., 2019), DIGL (Klicpera et al., 2019), ADGN (Sun & Wu, 2020) and SDRF (Topping et al., 2022)) (lower half of Table 1). The classical GIN model is used as the backbone for FireGNN and DE-GNN (Xu et al., 2019). All devised counterparts and selected variants other than FireGNN cannot be successfully applied to the resilience task in an inductive fashion, when only topological structures are available. The ResiNet's training setup is detailed in Appendix C.2. All algorithms are repeated for 3 random seeds using default hyper-parameters.

**Metrics.** Following the conventional setting in network science, resilience metrics used in our experiments include graph connectivity-based (Schneider et al., 2011) and spectrum-based measurements (adjacency matrix spectrum and Laplacian matrix spectrum). Utility metrics consist of global efficiency and local efficiency (Latora & Marchiori, 2003; Boccaletti et al., 2006). The detailed definitions are deferred to Appendix B.

## 5.2 COMPARISONS TO THE BASELINES

In this section, we compare ResiNet to baselines in optimizing the combination of resilience and utility with weight coefficient $\alpha \in \{0, 0.5\}$. Following conventional setting, the graph connectivity-

Table 1: Resilience optimization algorithm under the fixed maximal rewiring number budget of 20. Entries are in the format of $X(Y)$, where 1) $X$: weighted sum of the graph connectivity-based resilience and the network efficiency improvement (in percentage); 2) $Y$: required rewiring number.

| **Method** | $\alpha$ | BA-15 | BA-50 | BA-100 | BA-500 | BA-1000 | EU | P2P-Gnutella05 | P2P-Gnutella09 |
|---|---|---|---|---|---|---|---|---|---|
| HC | 0 | 26.8 (10) | 30.0 (20) | 24.1 (20) | 6.4 (20) | 66.6 (20) | 19.8 (20) | 6.2 (20) | 8.4 (20) |
| | 0.5 | 18.6 (11.3) | 22.1 (20) | 14.9 (20) | 5.9 (20) | 16.4 (20) | 16.3 (20) | 5.2 (20) | 7.0 (20) |
| SA | 0 | 21.6 (17.3) | 11.9 (20) | 12.5 (20) | 3.8 (20) | 42.9 (20) | 14.9 (20) | 3.9 (20) | 3.7 (20) |
| | 0.5 | 16.8 (19.0) | 11.4 (20) | 13.4 (20) | 4.0 (20) | 15.4 (20) | 14.0 (20) | 6.3 (20) | 4.8 (20) |
| Greedy | 0 | 23.5 (6) | 48.6 (13) | 64.3 (20) | ✗ | ✗ | 0.5 (3) | ✗ | ✗ |
| | 0.5 | 5.3 (15) | 34.7 (13) | 42.7 (20) | ✗ | ✗ | 0.3 (3) | ✗ | ✗ |
| EA | 0 | 8.5 (20) | 6.4 (20) | 4.0 (20) | 8.5 (20) | **174.1** (20) | 8.2 (20) | 2.7 (20) | 0 (20) |
| | 0.5 | 6.4 (20) | 4.7 (20) | 2.8 (20) | 5.6 (20) | 18.7 (20) | 9.3 (20) | 3.7 (20) | 0.1 (20) |
| DE-GNN-RL | 0 | 13.7 (2) | 0 (1) | 0 (1) | 1.6 (20) | 41.7 (20) | 9.0 (20) | 2.2 (20) | 0 (1) |
| | 0.5 | 10.9 (2) | 0 (1) | 0 (1) | 2.7 (20) | 20.1 (14) | 2.1 (20) | 0 (1) | 1.0 (20) |
| $k$-GNN-RL | 0 | 13.7 (2) | 0 (1) | 0 (1) | 0 (1) | 8.8 (20) | 4.5 (20) | -0.2 (20) | 0 (1) |
| | 0.5 | 6.3 (2) | 0 (1) | 0 (1) | 0 (20) | -24.9 (20) | 4.8 (20) | -0.1 (20) | 0 (1) |
| DIGL-RL | 0 | 9.8 (2) | 0 (1) | 0 (1) | ✗ | ✗ | 5.9 (20) | ✗ | ✗ |
| | 0.5 | 6.3 (2) | 0 (1) | 0 (1) | ✗ | ✗ | 7.2 (20) | ✗ | ✗ |
| AGDN-RL | 0 | 7.8 (4) | 0 (1) | 0 (1) | ✗ | ✗ | 0 (20) | ✗ | ✗ |
| | 0.5 | 1.4 (2) | 0 (1) | 0 (1) | ✗ | ✗ | 0.9 (20) | ✗ | ✗ |
| SDRF-RL | 0 | 9.8 (2) | 0 (1) | 0 (1) | ✗ | ✗ | 4.7 (20) | ✗ | ✗ |
| | 0.5 | 8.0 (2) | 0 (1) | -4.7 (20) | ✗ | ✗ | 5.3 (20) | ✗ | 55 |
| ResiNet | 0 | **35.3** (6) | **61.5** (20) | **70.0** (20) | **10.2** (20) | 172.8 (20) | **54.2** (20) | **14.0** (20) | **18.6** (20) |
| | 0.5 | **26.9** (20) | **53.9** (20) | **53.1** (20) | **15.7** (20) | **43.7** (20) | **51.8** (20) | **12.4** (20) | **15.1** (20) |

based metric is used as resilience metric (Schneider et al., 2011) and the global efficiency is used as utility metric (Latora & Marchiori, 2003; Boccaletti et al., 2006).

Table 1 records the metric gain and the required number of rewiring operations of different methods under the same rewiring budget. ResiNet outperforms all baselines consistently on all datasets. Note that this performance may be achieved by ResiNet under a much fewer number of rewiring operations, such as on BA-15 with $\alpha = 0$. In contrast, despite approximately searching for all possible new edges, the greedy algorithm is trapped in a local optimum (as it maximizes the one-step resilience gain) and is too expensive to optimize the resilience of a network with more than 300 nodes. For SA, the initial temperature and the temperature decay rate need to be carefully tuned for each network. EA performs suboptimally with a limited rewiring budget due to the numerous rewiring operations required in the internal process (e.g., the crossover operator). Learning-based methods using existing GNNs coupled with distance encoding cannot learn effectively compared to ResiNet, supporting our claim about the effectiveness of FireGNN on graphs without rich features.

### 5.3 Ablation study of ResiNet

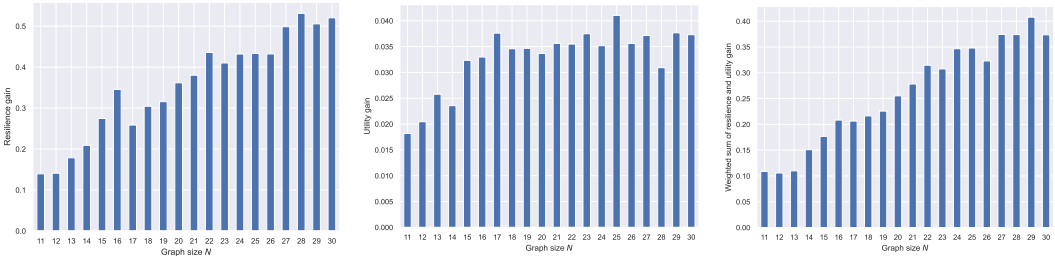
In this section, we investigate the impact of coefficient $\alpha$ of the objective on ResiNet and the effect of the filtration order $K$ on FireGNN.

To investigate the impact of the $\alpha$ in the reward function on ResiNet, we run a grid search by varying $\alpha$ from 0 to 1 and summarize the resilience gain, utility gain and the sum of them in Table 2. Table 2 shows that when we only optimize the resilience with $\alpha = 0$, the utility will degrade. Similarly, the resilience would also decrease if we only optimize the utility with $\alpha = 1$. This suggests a general tradeoff between resilience and utility and is consistent with their definitions. However, despite this tradeoff, we can achieve resilience gain and utility gain simultaneously on BA-15 and BA-50 since the original graph usually does not have the maximum resilience or utility. This incentivizes almost every network to conduct such optimization to some extent when feasible.

In FireGNN, the filtration order $K$ of FireGNN determines the total number of subgraphs involved in calculating the final node embedding, edge embedding, and graph embedding. FireGNN degenerates to existing GNNs when the filtration order $K$ is 0. Table 1 validates the effectiveness and necessity of FireGNN. Without FireGNN (other GNNs as the backbone), it is generally challenging for ResiNet to find a positive gain on graphs without rich features since ResiNet cannot learn to select the correct edges with the incorrect edge embeddings. The maximum $K$ of each dataset is recorded in Appendix Table 7, which shows that the maximum $K$ equals the around half size of the graph since we gradually remove the node with the largest degree, leading to a fast graph filtration process. For our experiments, we use the maximum of $K$ for graphs of sizes less than 50 and set $K$ = 3 (1) for graphs of sizes larger than 50 (200). To validate that ResiNet is not sensitive to $K$, we run a grid search on several datasets

Table 2: The effect of the coefficient $\alpha$ on ResiNet. The result is shown as percentages.

| Dataset | Gain | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BA-15 | Resilience | 35.3 | 35.3 | 35.3 | 33.3 | 17.6 | 17.6 | 27.5 | 17.6 | 17.6 | 17.6 | -2.0 |
| | Utility | -5.9 | -3.9 | -3.8 | -2.7 | 1.1 | 1.1 | 0 | 1.1 | 1.1 | 1.1 | 5.4 |
| | Reward | 35.3 | 34.2 | 32.9 | 29.7 | 15.2 | 14.2 | 19.7 | 11.4 | 9.2 | 6.0 | 5.4 |
| BA-50 | Resilience | 56.7 | 51.1 | 42.3 | 48.6 | 53.9 | 59.2 | 51.4 | 50.6 | 48.1 | 39.3 | -19.1 |
| | Utility | -3.6 | 3.4 | -2.1 | -4.0 | -4.2 | -4.2 | -2.6 | -2.2 | -2.1 | 0.5 | 5.5 |
| | Reward | 56.7 | 49.5 | 39.9 | 43.1 | 44.9 | 45.6 | 35.7 | 30.1 | 22.0 | 11.8 | 5.5 |
| BA-100 | Resilience | 75.4 | 74.6 | 74.8 | 76.1 | 72.8 | 72.8 | 75.1 | 75.4 | 74.9 | 71.6 | -11.8 |
| | Utility | -4.0 | -4.6 | -3.9 | -5.1 | -4.2 | -4.2 | -3.8 | -3.7 | -3.5 | -2.5 | 4.8 |
| | Reward | 75.4 | 71.9 | 69.0 | 66.4 | 59.4 | 54.3 | 49.7 | 41.8 | 31.1 | 16.7 | 4.8 |




(a) Inductivity on resilience (b) Inductivity on utility (c) Inductivity on both metrics

Figure 4: The inductive ability of ResiNet on the test dataset (BA-10-30) when optimizing (a) network resilience, (b) network utility, and (c) their combination.

to optimize the resilience by setting $K = 0, 1, 2, 3$. As shown in Appendix Table 5, the resilience is improved significantly with $K > 0$ and ResiNet performs well with $K = 1$ or $K = 2$.

## 6 Generalization

To demonstrate the inductivity of ResiNet, we first train ResiNet on two different datasets (BA-10-30 and BA-20-200) with the data setting listed in Appendix Table 3, and then evaluate its performance on an individual test dataset. The test dataset is not observed to ResiNet during the training process and fine-tuning is not allowed. We report the averaged resilience gain for the graphs of the same size for each dataset. The performance of ResiNet on BA-10-30 is shown in Figure 4 and the results of other datasets are deferred to Figure 8 in Appendix D. Figure 4 shows a nearly linear improvement of resilience with the increase of graph size, which is also consistent with the results in the transductive setting that larger graphs usually have a larger room to improve their resilience. Moreover, we conduct experiments to demonstrate ResiNet's generalization on optimizing different utility and resilience metrics, and the details are deferred to Appendix D.

## 7 Conclusion

In this work, we propose a general learning-based model, ResiNet, for the discovery of resilient network topologies with minimal changes to the graph structure. ResiNet is the *first* inductive method that formulates the task of boosting network resilience as an MDP of successive edge rewiring operations. Our technical innovation, FireGNN, as the graph feature extractor in ResiNet for handling graphs with only topologies available, is motivated by persistent homology. FireGNN alleviates the insufficiency of current GNNs (including GNNs more powerful than 1-WL test) on modeling graphs without rich features. By decomposing graphs into temporal subgraphs and learning to combine the individual representations from each subgraph, FireGNN can learn meaningful representations on the resilience task to provide sufficient gradients for training the RL agent while current GNNs fail. Our method is practically feasible as it balances the utility of the networks when boosting resilience. FireGNN is potentially general enough to be applied to solve various graph problems without rich features.

## REFERENCES


Mehmet E Aktas, Esra Akbas, and Ahmed El Fatmaoui. Persistence homology of networks: methods and applications. *Applied Network Science*, 4(1):1–28, 2019.

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 2002.

Réka Albert, István Albert, and Gary L Nakarado. Structural vulnerability of the north american power grid. *Physical review E*, 69(2):025103, 2004.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? A large-scale study. In *International Conference on Learning Representations*, 2021.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations*, 2016.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34, 2021.

Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(4-5):175–308, 2006.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F Montufar, and Michael Bronstein. Weisfeiler and lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, 2021.

Matteo Boffa, Zied Ben-Houidi, Jonatan Krolikowski, and Dario Rossi. Neural combinatorial optimization beyond the TSP: existing architectures under-represent graph structure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

Béla Bollobás and Oliver Riordan. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics*, 1(1):1–35, 2004.

Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations*, 2022.

Pierre Buesser, Fabio Daolio, and Marco Tomassini. Optimizing the robustness of scale-free networks with simulated annealing. In *International Conference on Adaptive and Natural Computing Algorithms*, pp. 167–176. Springer, 2011.

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. GraphNorm: A principled approach to accelerating graph neural network training. In *Proceedings of the International Conference on Machine Learning*, 2021.

Vincenza Carchiolo, Marco Grassia, Alessandro Longheu, Michele Malgeri, and Giuseppe Mangioni. Network robustness improvement via long-range links. *Computational Social Networks*, 6(1):1–16, 2019.

Hau Chan and Leman Akoglu. Optimizing network robustness by edge rewiring: a general framework. *Data Mining and Knowledge Discovery*, 30(5):1395–1425, 2016.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International Conference on Machine Learning*, 2018.

Herbert Edelsbrunner and John Harer. Persistent homology-A survey. *Contemporary Mathematics*, 453:257–282, 2008.

Changjun Fan, Li Zeng, Yizhou Sun, and Yang-Yu Liu. Finding key players in complex networks through deep reinforcement learning. *Nature Machine Intelligence*, pp. 1–8, 2020.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large TSP instances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Marco Grassia, Manlio De Domenico, and Giuseppe Mangioni. Machine learning dismantling and early-warning signals of disintegration in complex systems. *Nature Communications*, 12(1):1–10, 2021.

Xiaojie Guo, Lingfei Wu, and Liang Zhao. Deep graph translation. *CoRR*, abs/1805.09980, 2018. URL http://arxiv.org/abs/1805.09980.

Andrew G Haldane and Robert M May. Systemic risk in banking ecosystems. *Nature*, 469(7330):351–355, 2011.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *Proceedings of the International Conference on Machine Learning*, 2020.

Christoph Hofer, Florian Graf, Bastian Rieck, Marc Niethammer, and Roland Kwitt. Graph filtration learning. In *International Conference on Machine Learning*, 2020.

Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5):056109, 2002.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *CoRR*, abs/2006.14171, 2020. URL https://arxiv.org/abs/2006.14171.

Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. In *International Conference on Learning Representations*, 2022.

Swami Iyer, Timothy Killingback, Bala Sundaram, and Zhen Wang. Attack robustness and centrality of complex networks. *PloS One*, 8(4):e59613, 2013.

Daniel A Jaume, Adrián Pastine, and Victor Nicolas Schvöllner. 2-switch: transition and stability on graphs and forests. *arXiv preprint arXiv:2004.11164*, 2020.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *CoRR*, abs/1906.01227, 2019. URL http://arxiv.org/abs/1906.01227.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, pp. 1–29, 2022.

Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

P. Diederik Kingma and Lei Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, 2017.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, 2019.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

Vito Latora and Massimo Marchiori. Efficient behavior of small-world networks. *Physical Review Letters*, 87 (19):198701, 2001.

Vito Latora and Massimo Marchiori. Economic small-world behavior in weighted networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 32(2):249–263, 2003.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International conference on Knowledge Discovery and Data Mining*, 2006.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All you need to train deeper GCNs. *arXiv preprint arXiv:2006.07739*, 2020a.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *Advances in Neural Information Processing Systems*, 2020b.

Wenguo Li, Yong Li, Yi Tan, Yijia Cao, Chun Chen, Ye Cai, Kwang Y Lee, and Michael Pecht. Maximizing network resilience against malicious attacks. *Scientific Reports*, 9(1):1–9, 2019.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, 2018.

Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. Graph adversarial attack via rewiring. In *KDD*, pp. 1161–1169. ACM, 2021.

Sourav Medya, Tianyi Ma, Arlei Silva, and Ambuj Singh. A game theoretic approach for k-core minimization. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems*, 2020.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Damon Mosk-Aoyama. Maximum algebraic connectivity augmentation is NP-hard. *Operations Research Letters*, 36(6):677–679, 2008.

Adilson E Motter, Natali Gulbahce, Eivind Almaas, and Albert-László Barabási. Predicting synthetic rescues in metabolic networks. *Molecular systems biology*, 4(1):168, 2008.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 2018.

Bo Peng, Jiahai Wang, and Zizhen Zhang. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. *CoRR*, abs/2002.03282, 2020. URL https://arxiv.org/abs/2002.03282.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: graph contrastive coding for graph neural network pre-training. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2020.

Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *arXiv preprint cs/0209028*, 2002.

Lei Rong and Jing Liu. A heuristic algorithm for enhancing the robustness of scale-free networks based on edge classification. *Physica A: Statistical Mechanics and its Applications*, 503:503–515, 2018.

Christian M Schneider, André A Moreira, José S Andrade, Shlomo Havlin, and Hans J Herrmann. Mitigation of malicious attacks on networks. *Proceedings of the National Academy of Sciences*, 108(10):3838–3841, 2011.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Ricard V Sole and Ma Montoya. Complexity and fragility in ecological networks. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1480):2039–2045, 2001.

Chuxiong Sun and Guoshi Wu. Adaptive graph diffusion networks with hop-wise attention. *arXiv preprint arXiv:2012.15024*, 2020.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Rakshit Trivedi, Jiachen Yang, and Hongyuan Zha. GraphOpt: Learning optimization models of graph formation. In *International Conference on Machine Learning*, pp. 9603–9613. PMLR, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.

Kazuyoshi Wakuta. Vector-valued markov decision processes and the systems of linear inequalities. *Stochastic Processes and Their Applications*, 56(1):159–169, 1995.

Xiangrong Wang, Evangelos Pournaras, Robert E Kooij, and Piet Van Mieghem. Improving robustness of complex networks via the effective graph resistance. *The European Physical Journal B*, 87(9):1–12, 2014.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

A Yasin Yazıcıoğlu, Magnus Egerstedt, and Jeff S Shamma. Formation of robust multi-agent networks through self-organizing random regular graphs. *IEEE Transactions on Network Science and Engineering*, 2(4): 139–151, 2015.

Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, et al. Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, 2018.

Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems*, 2020.

James Jian Qiao Yu, Wen Yu, and Jiatao Gu. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3806–3817, 2019. doi: 10.1109/TITS.2019.2909109. URL https://doi.org/10.1109/TITS.2019.2909109.

Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Deep latent graph matching. In *International Conference on Machine Learning*, 2021.

Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

Kangfei Zhao, Zhiwei Zhang, Yu Rong, Jeffrey Xu Yu, and Junzhou Huang. Finding critical users in social communities via graph convolutions. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021. doi: 10.1109/TKDE.2021.3089763.

Mingxing Zhou and Jing Liu. A memetic algorithm for enhancing the robustness of scale-free networks against malicious attacks. *Physica A: Statistical Mechanics and its Applications*, 410:131–143, 2014.

Qiong Zhou and Janusz W Bialek. Approximate model of european interconnected system as a benchmark system to study effects of cross-border trades. *IEEE Transactions on Power Systems*, 20(2):782–788, 2005.

Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. A survey on graph structure learning: Progress and opportunities. In *IJCAI*, 2021.

# APPENDIX

## A EXTENDED RELATED WORK

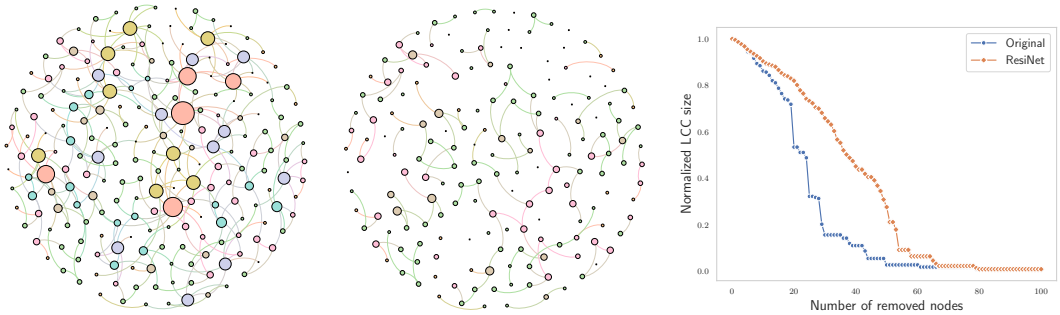
**Network resilience and utility.** Network utility refers to the system's quality to provide a specific service, for example, transmitting electricity in power networks and transmitting packages in routing networks. A popular metric for network utility is the network efficiency (Latora & Marchiori, 2003; Boccaletti et al., 2006). Network resilience measures the ability to prevent utility loss under failures and attacks. In many previous work, despite that network resilience could be improved, the utility may dramatically drop at the same time (Li et al., 2019; Carchiolo et al., 2019; Wang et al., 2014; Schneider et al., 2011; Chan & Akoglu, 2016; Buesser et al., 2011). This contradicts the idea behind improving network resilience and will be infeasible in real-world applications. Our goal is to enhance network resilience with moderate loss of network utility by network structure manipulations.

**Multi-views graph augmentation for GNNs.** Multi-views graph augmentation is one efficient way to improve the expressive power of GNNs or combine domain knowledge, which is adapted based on the task's prior (Hu et al., 2020). For example, GCC generates multiple subgraphs from the same ego network (Qiu et al., 2020). DGI maximizes the mutual information between global and local information (Velickovic et al., 2019). GCA adaptively incorporates various priors for topological and semantic aspects of the graph (You et al., 2020). (Hassani & Khasahmadi, 2020) contrasts representations from first-order neighbors and a graph diffusion. DeGNN Jin et al. (2020) was proposed as an automatic graph decomposition algorithm to improve the performance of deeper GNNs. These techniques rely on the existence of rich graph feature and the resultant GNNs cannot work well on graphs without rich features. In the resilience task, only the graph topological structure is available. Motivated by the calculation process of persistent homology (Edelsbrunner & Harer, 2008), we apply the filtration process to enhance the expressive power of GNNs for handling graphs without rich features.

**Deep graph generation.** Deep graph generation models learn the distribution of given graphs and generate more novel graphs. Some work use the encoder-decoder framework by learning latent representation of the input graph through the encoder and then generating the target graph through the decoder. For example, GCPN (You et al., 2018) incorporates chemistry domain rules on molecular graph generation. GT-GAN (Guo et al., 2018) proposes a GAN-based model on malware cyber-network synthesis. GraphOpt (Trivedi et al., 2020) learns an implicit model to discover an underlying optimization mechanism of the graph generation using inverse reinforcement learning. GFlowNet learns a stochastic policy for generating molecules with the probability proportional to a given reward based on flow networks and local flow-matching conditions Bengio et al. (2021). Boosting network resilience in a degree-preserving way can be viewed as a constrained graph generation task. However, constrained version of graph generation is still under development and none of existing methods can generate desired graphs with the exact node degree preserving constraint, which is required by the resilience task.

**Graph rewiring.** Graph rewiring aims to build novel classes of GNNs by preprocessing a given graph to overcome the problems of the over-squashing issue of training GNNs. For example, Graph diffusion convolution (GDC) was proposed to improve GNN's performance on downstream tasks by replacing message passing with graph diffusion convolution (Klicpera et al., 2019). Although conceptually related, graph rewiring differs from the edge rewiring operation used in our paper since edge rewiring can preserve node degree. An edge-based combinatorial curvature was proposed in (Topping et al., 2022) to help alleviate the over-squashing phenomenon in GNNs. To the best of our knowledge, currently, there is no inductive learning-based graph rewiring method. Moreover, graph rewiring methods rely on the existence of rich features to train GNNs on downstream tasks.

**Graph structure learning.** Unlike the graph generation task which focuses on the quality of the generated graph, graph structure learning (GSL) aims to jointly learn an optimized graph structure and corresponding graph representations only for better performance on downstream tasks. Although conceptually related, GSL differs from graph generation since GSL mostly cares for the downstream task while graph generation focuses on the generated graphs (Jin et al., 2020; Zhu et al., 2021). Currently, GSL relies on the existence of rich features to construct a graph, while there are generally




(a) Original EU network (b) EU network after attacks (c) Remaining LCC size under attack

Figure 5: The EU power network under the adaptive degree-based attack with (a) original EU network with 217 nodes, (b) remaining EU network after a series of attacks on 40 nodes, and (c) normalized size of the largest connected component (LCC).

no rich features in graph generation. Moreover, GSL cannot control the graph's node degree during graph optimization. We refer the interested readers to a survey of GSL (Zhu et al., 2021) since our work is a constrained graph generation task unrelated to GSL.

# B DEFINITIONS OF DIFFERENT OBJECTIVE FUNCTIONS

In this section, we present resilience definitions and utility definitions used in our experiments.

## B.1 RESILIENCE DEFINITIONS

Three kinds of resilience metrics are considered:

- The graph connectivity-based measurement is defined as (Schneider et al., 2011)

$$\mathcal{R}(G) = \frac{1}{N}\sum_{q=1}^{N} s(q)\,,$$

  where $s(q)$ is the fraction of nodes in the largest connected remaining graph after removing $q$ nodes from graph $G$ according to certain attack strategy. The range of possible values of $\mathcal{R}$ is $[1/N, 1/2]$, where these two extreme values correspond to a star network and a fully connected network, respectively. Figure 5 shows that the failures of a dozen of nodes could jeopardize the connectivity and utility of the EU power network.
- The spectral radius ($\mathcal{SR}$) denotes the largest eigenvalue $\lambda_1$ of an adjacency matrix.
- The algebraic connectivity ($\mathcal{AC}$) represents the second smallest eigenvalue of the Laplacian matrix of $G$.

## B.2 UTILITY DEFINITIONS

In this paper, the global and local communication efficiency are used as two measurements of the network utility, which are widely applied across diverse applications of network science, such as transportation and communication networks (Latora & Marchiori, 2003; Boccaletti et al., 2006).

The average efficiency of a network $G$ is defined as inversely proportional to the average over pairwise distances (Latora & Marchiori, 2001) as

$$E(G) = \frac{1}{N(N-1)}\sum_{i\neq j\in V}\frac{1}{d(i,j)}\,,$$

where $N$ denotes the total nodes in a network and $d(i,j)$ is the length of the shortest path between a node $i$ and another node $j$.

We can calculate the global and local efficiency given the average efficiency.

- The global efficiency of a network $G$ is defined as (Latora & Marchiori, 2001; 2003)

$$E_{global}(G) = \frac{E(G)}{E(G^{ideal})},$$

where $G^{ideal}$ is the "ideal" fully-connected graph on $N$ nodes and the range of $E_{global}(G)$ is [0, 1].

- The local efficiency of a network $G$ measures a local average of pairwise communication efficiencies and is defined as (Latora & Marchiori, 2001)

$$E_{local}(G) = \frac{1}{N}\sum_{i \in V} E(G_i),$$

where $G_i$ is the local subgraph including only of a node $i$'s one-hop neighbors, but not the node $i$ itself. The range of $E_{local}(G)$ is [0, 1].

# C IMPLEMENTATION DETAILS OF RESINET

This section provides the implementation details of ResiNet, including dataset, network structure training strategies, and node feature construction.

## C.1 DATASET

We first present the data generation strategies. Table 3 summarizes the statistics of each dataset. Synthetic datasets are generated using the Barabasi-Albert (BA) model (known as scale-free graphs) (Albert & Barabási, 2002), with the graph size varying from $|N|$=10 to $|N|$=1000. During the data generation process, each node is connected to two existing nodes for graphs with no more than 500 nodes, and each node is connected to one existing node for graphs with near 1000 nodes. BA graphs are chosen since they are vulnerable to malicious attacks and are commonly used to test network resilience optimization algorithms (Bollobás & Riordan, 2004). We test the performance of ResiNet on both transductive and inductive settings.

- **Transductive setting.** The algorithm is trained and tested on the same network.
  - Randomly generated synthetic BA networks, denoted by BA-$m$, are adopted to test the performance of ResiNet on networks of various sizes, where $m \in \{15, 50, 100, 500, 1000\}$ is the graph size.
  - The Gnutella peer-to-peer network file sharing network from August 2002 (Leskovec et al., 2007; Ripeanu et al., 2002) and the real EU power network (Zhou & Bialek, 2005) are used to validate the performance of ResiNet on real networks. The random walk sampling strategy is used to derive a representative sample subgraph with hundreds of nodes from the Gnutella peer-to-peer network (Leskovec & Faloutsos, 2006).
- **Inductive setting.** Two groups of synthetic BA networks denoted by BA-$m$-$n$ are randomly generated to test ResiNet's inductivity, where $m$ is the minimal graph size, and $n$ indicates the maximal graph size. We first randomly generate the fixed number of BA networks as the training data to train ResiNet and then evaluate ResiNet's performance directly on the test dataset without any additional optimization.

## C.2 RESINET SETUP

In this section, we provide detailed parameter setting and training strategies for ResiNet.

Our proposed FireGNN is used as the graph encoder in ResiNet, including a 5-layer defined GIN (Xu et al., 2019) as the backbone. The hidden dimensions for node embedding and graph embedding in each hidden layer are set to 64 and the SeLU activation function (Klambauer et al., 2017) is used after each message passing propagate. Graph normalization strategy is adopted to stabilize the training of GNN (Cai et al., 2021). The jumping knowledge network (Xu et al., 2018) is used to aggregate node features from different layers of the GNN.

Table 3: Statistics of graphs used for resilience maximization. Both transductive and inductive settings ($\star$) are included. Consistent with our implementation, we report the number of edges by transforming undirected graphs to directed graphs. The edge rewiring has a fixed execution order. For the inductive setting, we report the maximum number of edges. The action space size of the edge rewiring is measured by $2|E|^2$.

| Dataset | Node | Edge | Action Space Size | Train/Test | Setting |
|---|---|---|---|---|---|
| BA-15 | 15 | 54 | 5832 | ✗ | Transductive |
| BA-50 | 50 | 192 | 73728 | ✗ | Transductive |
| BA-100 | 100 | 392 | 307328 | ✗ | Transductive |
| BA-500 | 500 | 996 | 1984032 | ✗ | Transductive |
| BA-1000 | 1000 | 999 | 1996002 | ✗ | Transductive |
| EU | 217 | 640 | 819200 | ✗ | Transductive |
| p2p-Gnutella05 | 400 | 814 | 1325192 | ✗ | Transductive |
| p2p-Gnutella09 | 300 | 740 | 1095200 | ✗ | Transductive |
| BA-10-30 ($\star$) | 10-30 | 112 | 25088 | 1000/500 | Inductive |
| BA-20-200 ($\star$) | 20-200 | 792 | 1254528 | 4500/360 | Inductive |

The overall policy is trained by using the highly tuned implementation of proximal policy optimization (PPO) algorithm (Schulman et al., 2017). Several critical strategies for stabilizing and accelerating the training of ResiNet are used, including advantage normalization (Andrychowicz et al., 2021), the dual-clip PPO (the dual clip parameter is set to 10) (Ye et al., 2020), and the usage of different optimizers for policy network and value network. Additionally, since the step-wise reward range is small (around 0.01), we scale the reward by a factor of 10 to aim the training of ResiNet. The policy head model and value function model use two separated FireGNN encoder networks with the same architecture. ResiNet is trained using two separate Adam optimizers (Kingma & Ba, 2015) with batch size 256 and a linearly decayed learning rate of 0.0007 for the policy network and a linearly decayed learning rate of 0.0007 for the value network. The aggregation function of FireGNN is defined as an attention mechanism-based linear weighted combination.

**Hardware**: We run all experiments for ResiNet on the platform with two GEFORCE RTX 3090 GPU and one AMD 3990X CPU.

### C.3 Node feature construction

The widely-used node degree feature cannot significantly benefit the network resilience optimization of a single graph due to the degree-preserving rewiring. Therefore, we construct node features for each input graph to aid the transductive learning and inductive learning, including

- The distance encoding strategy (Li et al., 2020b). Node degree feature is a part of it.
- The 8-dimensional position embedding originating from the Transformer (Vaswani et al., 2017) as the measurement of the vulnerability of each node under attack. If the attack order is available, we can directly encode it into the position embedding. If the attack order is unknown, node degree, node betweenness, and other node priority metrics can be used for approximating the node importance in practice. In our experiments, we used the adaptive node degree for the position embedding.

### C.4 Baseline setup

All baselines share the same action space with ResiNet and use the same action masking strategy to block invalid actions as ResiNet does. The maximal objective evaluation is consistent for all algorithms. Other settings of baselines are consistent with the default values in their paper. The early-stopping strategy is used for baselines, which means that the search process terminates if no improvement is obtained in successive 1000 objective function calling trials.

Table 4: Performance gain (in percentage) of ResiNet in optimizing varying objectives on the BA-15 network. All objectives are optimized with the same hyper-parameters, which means that we did not tune hyper-parameters for objectives except for $R_D$.

| Objective | Gain (%) | Objective | Gain(%) |
|---|---|---|---|
| $\mathcal{R}_D$ | 35.3 | $\mathcal{R}_B$ | 14.6 |
| $\mathcal{SR}_D$ | 15.3 | $\mathcal{SR}_B$ | 15.3 |
| $\mathcal{AC}_D$ | 48.2 | $\mathcal{AC}_B$ | 43.2 |
| $\mathcal{R}_D + E_{global}$ | 14.2 | $\mathcal{R}_B + E_{global}$ | 13.1 |
| $\mathcal{SR}_D + E_{global}$ | 14.6 | $\mathcal{SR}_B + E_{global}$ | 15.0 |
| $\mathcal{AC}_D + E_{global}$ | 34.0 | $\mathcal{AC}_B + E_{global}$ | 31.3 |
| $\mathcal{R}_D + E_{local}$ | 24.4 | $\mathcal{R}_B + E_{local}$ | 39.4 |
| $\mathcal{SR}_D + E_{local}$ | 17.3 | $\mathcal{SR}_B + E_{local}$ | 21.2 |
| $\mathcal{AC}_D + E_{local}$ | 9.4 | $\mathcal{AC}_B + E_{local}$ | 15.1 |

# D EXTENDED EXPERIMENTAL RESULTS

In this section, we present additional experimental results to show that ResiNet generalizes to unseen graphs, different utility and resilience metrics.

## D.1 LEARNING TO BALANCE DIFFERENT UTILITY AND RESILIENCE METRICS

Figure 6 shows that ResiNet can accommodate various of $\alpha$ to balance the resilience and utility during the optimization process. As shown in Figure 7, we conduct extensive experiments on the BA-15 network to demonstrate that ResiNet can learn to optimize graphs with different resilience and utility metrics and to defend against other types of attacks besides the node degree-based attack, such as the node betweenness-based attack.

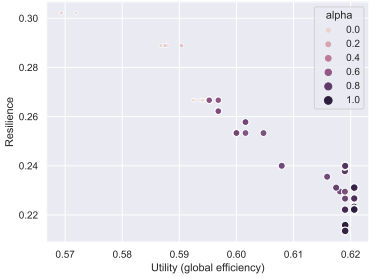



Figure 6: Pareto points obtained by ResiNet of balancing various combinations of resilience and utility.

Table 4 records the improvements in percentage of ResiNet for varying objectives on the BA-15 dataset. As visualized in Figure 7, ResiNet is not limited to defend against the node degree-based attack (Figure 7 (b)-(j)) and also learns to defend against the betweenness-based attack (Figure 7 (k)-(s)). Total three resilience metrics are used, with $\mathcal{R}$ denoting the graph connectivity-based resilience metric, $\mathcal{SR}$ being the spectral radius and $\mathcal{SR}$ representing the algebraic connectivity. Total two utility metrics are adopted, including the global efficiency $E_{global}$ and the local efficiency $E_{local}$. Not surprisingly, the optimized network with an improvement of about 3.6% for defending the betweenness-based attack also has a higher resilience (around 7.8%) against the node-degree attack. This may be explained as the similarity between node degree and betweenness for a small network.

## D.2 INDUCTIVITY ON LARGER DATASETS

To demonstrate that ResiNet can learn from networks to accommodate different utility and resilience metrics, we conduct experiments based on BA-15 using multiple resilience and utility metrics. The Pareto points shown in Figure 6 denote the optimum under different objectives on BA-15, implying that ResiNet can obtain the approximate Pareto frontier. Surprisingly, the initial gain of resilience (from around 0.21 to around 0.24) is obtained without loss of the utility, which incentivizes almost every network to conduct such optimization to some extent when feasible. More results are included in Appendix D.1 and the optimized network structures are visualized in Figure 7 and Figure 10.

Even with limited computational resources, armed with the autoregressive action space and the power of FireGNN, ResiNet can be trained fully end-to-end on graphs with thousands of nodes using RL. We demonstrate the inductivity of ResiNet on graphs of different sizes by training ResiNet on the BA-20-200 dataset, which consists of graphs with the size ranging from 20 to 200, and then report its performance on directly guiding the edges selections on unseen test graphs. The filtration order $K$ is

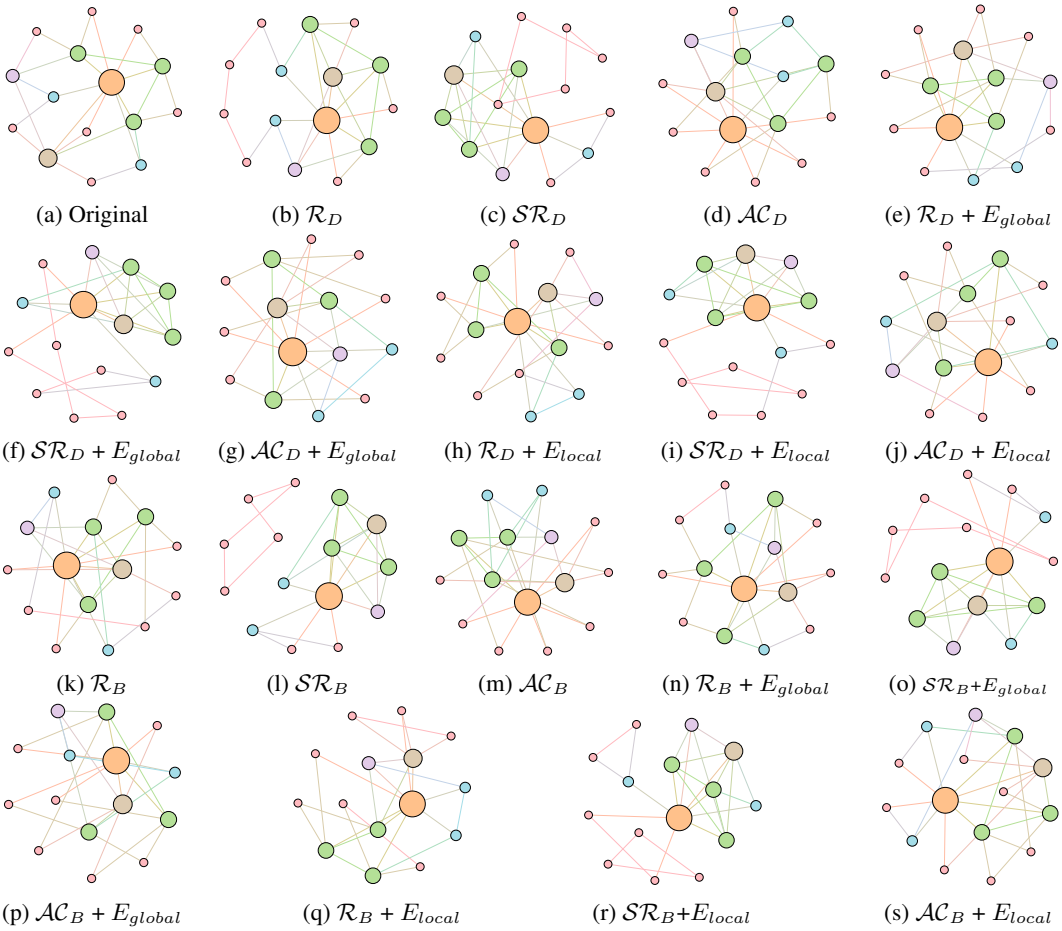


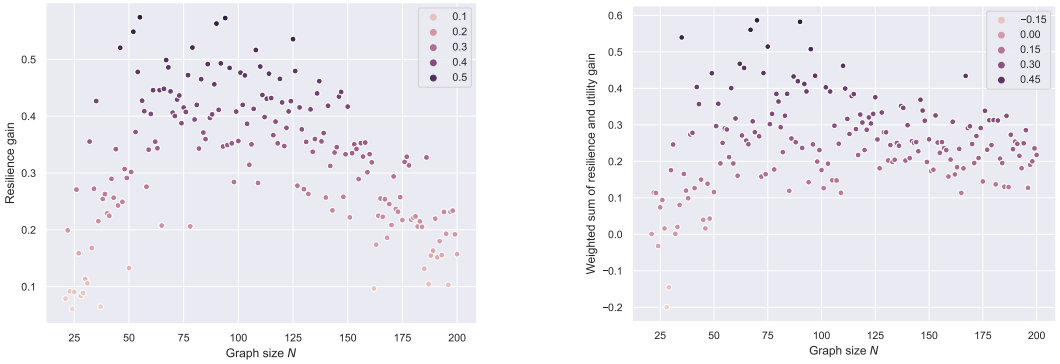

Figure 7: The resilience maximization on the BA-15 dataset with 15 nodes and 27 edges with (a) original network, (b)-(j) results of defending the node degree-based attack with different combinations of resilience and utility, and (k)-(s) results of defending against the node betweenness-based attack with varying combinations of resilience and utility. For three resilience metrics, $\mathcal{R}$ denotes the graph connectivity-based resilience metric; $\mathcal{SR}$ is the spectral radius; $\mathcal{SR}$ represents the algebraic connectivity. For two utility metrics, $E_{global}$ denotes the global efficiency, and $E_{local}$ is the local efficiency.

set to 1 for the computational limitation. As shown in Figure 8, we can see that ResiNet has the best performance for $N \in [70, 100]$. The degrading performance with the graph size may be explained by the fact that larger graphs require a larger filtration order for ResiNet to work well. A more stable performance improvement of ResiNet is observed with the increment of graph size when trained to optimize network resilience and utility simultaneously, and ResiNet possibly finds a strategy to balance these two metrics.

### D.2.1 The Effect of filtration order $K$ on FireGNN

In this section, we report the ratio of the remaining edges in subgraphs versus different filtration order $K$ in Table 6 and visualize it in Figure 9. The maximum filtration order $K$ of FireGNN of each dataset is summarized in Table 7. Table 7 demonstrates that the maximum filtration order $K$ is nearly around the half size of the graph since we gradually remove the node with the largest degree in the filtration process.

## D.3 Inspection of optimized networks




(a) Inductivity on resilience

(b) Inductivity on resilience and utility

Figure 8: The inductive ability of ResiNet on the test dataset (BA-20-200) when optimizing (a) network resilience and (b) the combination of resilience and utility.

Table 5: The effect of the filtration order $K$ on ResiNet in improving network resilience (percentage).

| $K$ | BA-15 | BA-50 | BA-100 | EU |
|---|---|---|---|---|
| 0 | 11.8± 2.7 | 0 ±0 | 0 ± 0 | 6.1 ± 4.2 |
| 1 | 17.6 ± 0 | 49.6 ± 2.3 | 74.9± 0.8 | 54.5 ± 0.4 |
| 2 | 17.6 ± 0 | 51.0 ± 0.1 | 76.3± 1.2 | 57.4 ± 1.6 |
| 3 | 17.6 ± 0 | 55.7±2.3 | 73.1 ± 0.9 | 54.9 ± 0.9 |

Moreover, to provide a deeper inspection into the optimized network structure, we take the EU power network as an example to visualize its network structure and the optimized networks given by ResiNet with different objectives. Compared to the original EU network, Figure 10 (b) is the network structure obtained by only optimizing the graph connectivity-based resilience. We can observe a more crowded region on the left, consistent with the "onion-like" structure concluded in previous studies. If we consider the combination gain of both resilience and utility, we observe a more compact clustering "crescent moon"-like structure as shown in Figure 10 (c).

### D.4 Performance comparisons under a large rewiring budget

In this section, we present the resilience improvement and the required number of edge rewiring of each algorithm under a large rewiring budget of 20. The running speed is also presented to compare the running time efficiency of each algorithm.

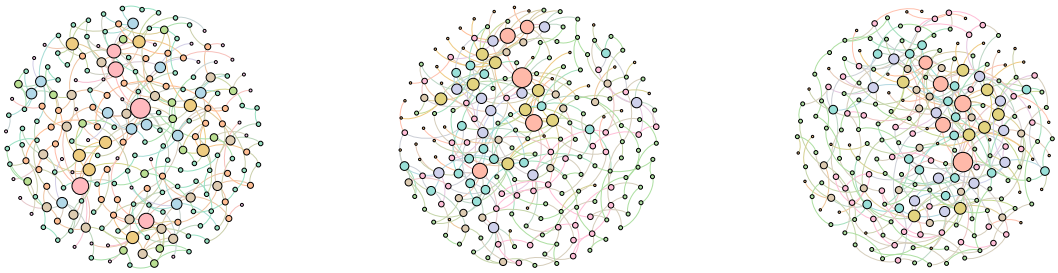
As shown in Table 8, traditional methods improve the network resilience significantly compared to ResiNet under a large rewiring budget of 200. However, traditional methods are still undesired in such a case since a solution with a large rewiring budget is not applicable in practice due to the vast cost of adding many new edges into a real system. For example, the actual number of rewiring budget for EA is hard to calculate since it is a population-based algorithm, so it is omitted in Table 8. All baselines adopt the early-stopping strategy that they will terminate if there is no positive resilience gain in a successive 1000 steps.

Table 6: The ratio of remaining edges in subgraphs versus the filtration order $K$.

| **K** | BA-15 | BA-50 | BA-100 | BA-500 | BA-1000 | EU | P2P-Gnutella05 | P2P-Gnutella09 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.6667 | 0.7708 | 0.8571 | 0.9538 | 0.9479 | 0.9625 | 0.9779 | 0.9608 |
| 2 | 0.4815 | 0.6667 | 0.7602 | 0.9096 | 0.9139 | 0.9313 | 0.957 | 0.927 |
| 3 | 0.3333 | 0.5938 | 0.6837 | 0.8665 | 0.8809 | 0.9031 | 0.9361 | 0.8973 |
| 4 | 0.2222 | 0.5208 | 0.6224 | 0.8313 | 0.8559 | 0.8781 | 0.9152 | 0.8703 |
| 5 | 0.1481 | 0.4583 | 0.5663 | 0.8002 | 0.8348 | 0.8562 | 0.8956 | 0.8446 |
| 6 | 0.0370 | 0.4062 | 0.5255 | 0.7741 | 0.8158 | 0.8344 | 0.8771 | 0.8189 |
| 7 | 0 | 0.3646 | 0.4847 | 0.754 | 0.7998 | 0.8125 | 0.8587 | 0.7959 |
| 8 | 0 | 0.3229 | 0.4439 | 0.7359 | 0.7848 | 0.7906 | 0.8403 | 0.773 |
| 9 | 0 | 0.2917 | 0.4031 | 0.7199 | 0.7698 | 0.7719 | 0.8231 | 0.75 |

Table 7: Maximum filtration order $K$ of each dataset.

| **Dataset** | BA-15 | BA-50 | BA-100 | BA-500 | BA-1000 | EU | P2P-Gnutella05 | P2P-Gnutella09 |
|---|---|---|---|---|---|---|---|---|
| Size (node) | 15 | 50 | 100 | 500 | 1000 | 217 | 400 | 300 |
| Maximum of K | 7 | 23 | 42 | 226 | 306 | 103 | 216 | 160 |



(a) Original EU network (b) $\mathcal{R}$ (c) $E_{global}$

Figure 10: Visualizations of the original EU network and optimized networks using ResiNet with different objectives: $\mathcal{R}$ means the connectivity-based resilience measurement and $E_{global}$ is the global efficiency.

Table 9 indicates that the time it takes for the benchmark algorithm to solve the problem usually increases as the test data set size increases. In contrast, our proposed ResiNet is suitable for testing on a large dataset once trained.

## E Deep analysis of why regular GNNs fail in the resilience task

It is well-known that GNNs generally work well for graphs with rich features. Unluckily, the graph network in the resilience task has no node/edge/graph feature, with only the topological structure available. No rich feature means that the output of the GNNs is not distinguishable, and then it is difficult for the RL agent to distinguish different vertices/edges, causing large randomness in the output of the policy. This

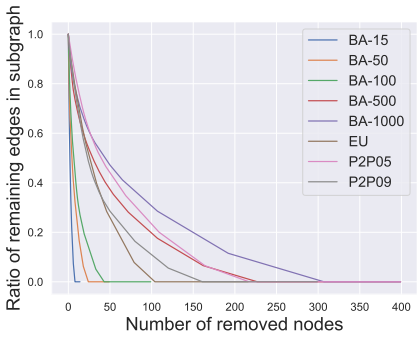



Figure 9: Ratio of the remaining edges in subgraphs versus different filtration order $K$

Table 8: Resilience optimization algorithm under the fixed maximal rewiring number budget of 200. Entries are in the format of $X(Y)$, where 1) $X$: weighted sum of the graph connectivity-based resilience and the network efficiency improvement (in percentage); 2) $Y$: required rewiring number. Results are averaged over 3 runs and best performance is in bold.

| Method | $\alpha$ | BA-15 | BA-50 | BA-100 | BA-500 | BA-1000 | EU | P2P-Gnutella05 | P2P-Gnutella09 |
|---|---|---|---|---|---|---|---|---|---|
| HC | 0 | 26.8 (10.0) | 52.1 (47.0) | 76.9 (97.3) | 45.8 (200) | 302.5 (200) | 71.9 (152.7) | 37.5 (193.3) | 40.2 (137.7) |
| | 0.5 | 18.6 (11.3) | 43.1 (62.7) | 56.9 (121) | 30.0 (200) | 66.3 (200) | 63.2 (200) | 27.7 (200) | 34.7 (196.3) |
| SA | 0 | 26.8 (20) | 49.7 (59.0) | 84.5 (119.7) | 43.2 (200) | 271.8 (200) | 73.5(160.3) | 37.1 (200) | 37.2 (134) |
| | 0.5 | 17.8 (21) | 41.1 (79.7) | 57.7 (127.7) | 31.4 (200) | 64.9 (200) | 62.8 (200) | 37.1 (200) | 35.2 (200) |
| Greedy | 0 | 23.5 (6) | 48.6 (13) | 64.3 (20) | ✗ | ✗ | 0.5 (3) | ✗ | ✗ |
| | 0.5 | 5.3 (15) | 34.7 (13) | 42.7 (20) | ✗ | ✗ | 0.3 (3) | ✗ | ✗ |
| EA | 0 | 35.3 (✗) | 50.2 (✗) | 61.9 (✗) | 9.9 (200) | 174.1 (200) | 66.2 (✗) | 2.3 (200) | 0 (200) |
| | 0.5 | 27.1 (✗) | 38.3 (✗) | 46.6 (✗) | 6.8 (200) | 18.7 (200) | 58.4 (✗) | 3.2 (200) | 0 (200) |
| DE-GNN-RL | 0 | 13.7 (2) | 0 (1) | 0 (1) | 1.6 (20) | 41.7 (20) | 9.0 (20) | 2.2 (20) | 0 (1) |
| | 0.5 | 10.9 (2) | 0 (1) | 0 (1) | 2.7 (20) | 20.1 (14) | 2.1 (20) | 0 (1) | 1.0 (20) |
| $k$-GNN-RL | 0 | 13.7 (2) | 0 (1) | 0 (1) | 0 (1) | 8.8 (20) | 4.5 (20) | -0.2 (20) | 0 (1) |
| | 0.5 | 6.3 (2) | 0 (1) | 0 (1) | 0 (20) | -24.9 (20) | 4.8 (20) | -0.1 (20) | 0 (1) |
| ResiNet | 0 | **35.3** (6) | **61.5** (20) | **70.0** (20) | **10.2** (20) | 172.8 (20) | **54.2** (20) | **14.0** (20) | **18.6** (20) |
| | 0.5 | **26.9** (20) | **53.9** (20) | **53.1** (20) | **15.7** (20) | **43.7** (20) | **51.8** (20) | **12.4** (20) | **15.1** (20) |

Table 9: Running speed (in second) of the resilience optimization algorithm under the fixed maximal rewiring number budget. Entries are in the format of $X(Y)$, where 1) $X$: speed under the budget of 20; 2) $Y$: speed under the budget of 200 . ✗ means that the result is not available at a reasonable time. Results are averaged over 3 runs and best performance is in bold.

| Method | $\alpha$ | BA-15 | BA-50 | BA-100 | BA-500 | BA-1000 | EU | P2P-Gnutella05 | P2P-Gnutella09 |
|---|---|---|---|---|---|---|---|---|---|
| HC | 0 | 1.0 (1.0) | 1.1 (6.4) | 1.3 (22.2) | 21.9 (354.1) | 80.3 (1288.3) | 3.1 (94.2) | 15.3 (358.1) | 4.5 (89.1) |
| | 0.5 | 1.5 (11.5) | 1.1 (12.8) | 2.0 (49.0) | 40.9 (589.5) | 148.7 (2603.7) | 5.3 (193.7) | 24.7 (462.8) | 7.0 (190.6) |
| SA | 0 | 0.5 (0.5) | 0.3 (6.6) | 0.6 (22.6) | 12.2 (313.0) | 45.7 (1051.8) | 2.4 (91.2) | 10.8 (286.4) | 2.6 (89.4) |
| | 0.5 | 0.7 (1.7) | 0.7 (13.2) | 1.7 (47.5) | 33.9 (568.9) | 99.8 (2166.3) | 5.0 (193.5) | 23.9 (454.5) | 6.3 (188.5) |
| Greedy | 0 | 0.2 (6.0) | 34.1 (34.5) | 766.3 (✗) | ✗ | ✗ | 3061.7 (✗) | ✗ | ✗ |
| | 0.5 | 0.7 (0.7) | 64.1 (65.4) | 1478.9 (✗) | ✗ | ✗ | 6192.6 (✗) | ✗ | ✗ |
| EA | 0 | 0.01 (✗) | 0.1 (✗) | 1.6 (✗) | 2.5 (✗) | 10.3 (✗) | 0.2 (✗) | 1.6 (✗) | 0.4 (✗) |
| | 0.5 | 0.01 (✗) | 0.1 (✗) | 0.8 (✗) | 4.7 (✗) | 15.0 (✗) | 0.4 (✗) | 3.0 (✗) | 0.8 (✗) |
| DE-GNN-RL | 0 | 0.1 (✗) | 0.1 (✗) | 0.1 (✗) | 14.9 (✗) | 70.3 (✗) | 3.6 (✗) | 8.7 (✗) | 0.5 (✗) |
| | 0.5 | 0.1 (✗) | 0.1 (✗) | 0.2 (✗) | 13.7 (✗) | 60.9 (✗) | 4.5 (✗) | 1.0 (✗) | 6.7 (✗) |
| $k$-GNN-RL | 0 | 0.02 (✗) | 0.03 (✗) | 0.07 (✗) | 1.3 (✗) | 56.5 (✗) | 2.6 (✗) | 8.2 (✗) | 0.5 (✗) |
| | 0.5 | 0.02 (✗) | 0.04 (✗) | 0.08 (✗) | 18.3 (✗) | 76.1 (✗) | 3.6 (✗) | 11.5 (✗) | 0.6 (✗) |
| ResiNet | 0 | 0.5 (✗) | 1.8 (✗) | 2.2 (✗) | 17.5 (✗) | 66.8 (✗) | 4.5 (✗) | 14.7 (✗) | 9.3 (✗) |
| | 0.5 | 0.5 (✗) | 1.9 (✗) | 2.4 (✗) | 18.0 (✗) | 67.5 (✗) | 5.2 (✗) | 15.0 (✗) | 10.3 (✗) |

Table 10: Characteristics of different graph related tasks.

| **Approach** | **Task** | **RL component** | | | **Problem** | **Training & Inference** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **State** | **Action** | **Reward** | **Complexity** | **Size Extrapolate** | **Encoder** | **Action Masking** | **Scalability** |
| S2V-DQN Khalil et al. (2017) | MVC | node level | add node to subset | -1 | $\mathcal{O}(N)$ | ✗ | S2V | ✓ | 500 |
| | Max-Cut | node level | add node to subset | change in cut weight | $\mathcal{O}(N)$ | ✗ | S2V | ✓ | 300 |
| | TSP | node level | add node to tour | change in tour cost | $\mathcal{O}(N)$ | ✗ | S2V | ✓ | 300 |
| Local search (Hudson et al., 2022) | TSP | edge level | relocate node in tour | global regret | $\mathcal{O}(N^2)$ | ✓ | GNN | ✓ | 100 |
| RNN-RL (Bello et al., 2016) | TSP | node level | add node to tour | change in tour cost | $\mathcal{O}(N)$ | ✗ | RNN | ✓ | 100 |
| GNN-RL (Joshi et al., 2022) | TSP | node level | add node to tour | change in tour cost | $\mathcal{O}(N)$ | ✓ | GNN | ✓ | 50 |
| Attention-RL (Kool et al., 2018) | TSP | node level | add node to tour | change in tour cost | $\mathcal{O}(N)$ | ✗ | Attention | ✓ | 100 |
| | VRP | node level | add node to tour | change in tour cost | $\mathcal{O}(N)$ | ✗ | Attention | ✓ | 100 |
| Local search (Böther et al., 2022) | MIS | node level | add node to subset | change in IS size | $\mathcal{O}(N)$ | ✓ | GNN | ✓ | 800 |
| **ResiNet** | Resilience | ***graph level*** | ***edge rewiring*** | change in resilience and utility | $\mathcal{O}(N^4)$ | ✓ | ***FireGNN*** | ✗ | ***1000*** |

may cause the rewiring process to alternate between
two graphs, forming an *infinite loop*. And we suspect that this infinite loop failure may explain the poor empirical performance of optimizing network resilience by selecting edges using existing GNNs and reinforcement learning (RL). The infinite loop failure is presented as follows.

Consider the graph $G_t$ with $N$ nodes and containing two edges $AB$ and $CD$. The agent selects $AB$ and $CD$ for rewiring, leading to $G_{t+1}$ with news edges $AC$ and $BD$. A frequent empirical failure of regular GNNs for the resilience task is the *infinite loop* phenomenon. The agent would select $AC$ and $BD$ at step $t+1$, returning back to $G_t$ and forming a cycled loop between $G_t$ and $G_{t+1}$. Formally, the infinite loop is formulated as

$$((A,B),(C,D)) = \underset{i,j,m,n \in 1:N}{\operatorname{argmax}} \ \mathrm{SIM}\left(((h_t^i, h_t^j),(h_t^m, h_t^n)), h_{G_t}\right)$$

$$((A,C),(B,D)) = \underset{i,j,m,n \in 1:N}{\operatorname{argmax}} \ \mathrm{SIM}\left(((h_{t+1}^i, h_{t+1}^j),(h_{t+1}^m, h_{t+1}^n)), h_{G_{t+1}}\right) ,$$

where SIM is a similarity metric, $h_t^i$ and $h_{G_t}$ are embeddings of node $i$ and graph $G_t$ at step $t$, and ($A$,$B$) is one edge.

Table 10 compares and summarizes different graph related tasks' characteristics. We can see that *the resilience task is more challenging from many aspects*. No prior rule like action masking or negative penalty can be used to avoid selecting visited nodes as in TSP. For the resilience task, all previously visited edges are also possibly valid to be selected again, resulting in insufficient training signals.

The desired GNN model should not depend on rules like action masking to distinguish edge and graph representations for graphs with little node features. Our proposed FireGNN fulfills these requirements to obtain proper training signals. FireGNN has a distinct expressive power and learns to create more meaningful and distinguishable features for each edge. FireGNN is not a simple aggregation of higher-order information of a static graph. It was inspired by homology filtration and the multi-view graph augmentation. Persistence homology motivates us to aggregate more distinct node features by observing how the graph evolves towards being empty, leading to more distinct and meaningful features for each node/edge, thus avoiding the infinite loop. Extensive experimental results in Table 1 validate the necessity and effectiveness of FireGNN. Existing GNNs perform worse while FireGNN performs well.