DESIGNING GRAPH NEURAL NETWORKS TRAINING DATA WITH LIMITED SAMPLES AND SMALL NETWORK SIZES

Junior Momo Ziazet[1], Charles Boudreau[1], Oscar Delgado[1], Brigitte Jaumard[1]
[1]Computer Science and Software Engineering, Concordia University, Montreal, Canada

NOTE: Corresponding author: Brigitte Jaumard, brigitte.jaumard@concordia.ca

Abstract – *Machine learning is a data-driven domain, which means a learning model's performance depends on the availability of large volumes of data to train it. However, by improving data quality, we can train effective machine learning models with little data. This paper demonstrates this possibility by proposing a methodology to generate high-quality data in the networking domain. We designed a dataset to train a given Graph Neural Network (GNN) that not only contains a small number of samples, but whose samples also feature network graphs of a reduced size (10-node networks). Our evaluations indicate that the dataset generated by the proposed pipeline can train a GNN model that scales well to larger networks of 50 to 300 nodes. The trained model compares favorably to the baseline, achieving a mean absolute percentage error of 5-6%, while being significantly smaller at 90 samples total (vs. thousands of samples for the baseline).*

Keywords – Data-centric AI, data generation, graph neural networks, network modeling, RouteNet-Fermi

1. INTRODUCTION

Recently, the data networking community has developed robust Graph Neural Networks (GNNs) able to accurately estimate complex network parameters with good accuracy [1]. These models were trained on large datasets, however, and in many networking scenarios data is not readily available or is prohibitively expensive to acquire. Therefore, being able to effectively train deep neural networks using smaller datasets becomes crucial, urging a shift from model-centric AI to data-centric AI.

Data-centric AI is an emerging field of study that aims to elaborate techniques for dataset optimization [2]. It proposes directing more focus towards developing systematic engineering practices to improve data quality, and away from architecture search and hyperparameter tuning. Hence, data-centric AI can reduce development time while improving model accuracy, promoting collaboration, and driving revenue [3].

To the best of our knowledge, in the networking domain, there is no public research work on how to produce good quality small datasets for training machine learning models. Inspired by the first "Data-Centric AI" competition that was organized by Andrew Ng *et al.* [4], the third edition of the "Graph Neural Network Challenge" [5] proposes to explore a data-centric AI approach for building accurate network digital twins. This paper summarizes our winning submission to the "Graph Neural Network Challenge 2022". It describes our methodology, presents the numerical results, and compares our solution with the other top solutions submitted to the competition.

The paper is structured as follows: Section 2 provides the problem description, followed by the state-of-the-art review in Section 3. Section 4 details the given GNN model, followed by Section 5 that describes our proposed solution. Section 6 explains our experiments and numerical results, and finally, we provide a summary of our conclusions in Section 7.

2. PROBLEM DESCRIPTION

This work covers our submission to the 2022 edition of the Graph Neural Networking Challenge. In the competition, we are given a state-of-the-art GNN model for network simulation (RouteNet-Fermi, [6]), and a packet-level simulator based on OMNET++ to generate datasets. The task consists of using data-centric AI principles [7] to produce a training dataset for the model. As data-centric AI emphasizes the quality of the data given to a model, we must improve the given model's performance without altering its architecture or hyper-parameters.

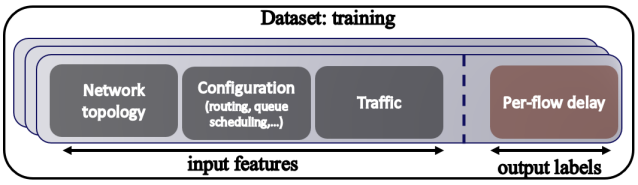

Fig. 1 – Components of a sample in the dataset

As described in [8], and represented in Fig. 1, one sample in the dataset consists of a tuple containing the following: a topology object, which holds data on the physical components (nodes and links) of the network, a routing matrix, which holds the unique path linking each node pair in the topology, and a traffic matrix, which contains information on the flows between each pair of nodes. The model takes these components as input features and output labels for the sample corresponding to the per-flow delay. All samples must be generated with the provided network simulator, and the provided neural network model. All the parameters for the learning process, i.e., model architecture (RouteNet-Fermi), learning rate (0.005), optimizer (Adam), loss function (mean absolute percentage error),

number of epoch (20 maximum), seed (69420), and the others training hyperparameters are fixed and given by the challenge organizers. This means that participants must not modify the learning parameters, and must concentrate solely on generating the dataset.

Data generated for the competition must meet certain requirements, which are summarized as follows. Created datasets are restricted to a small amount of samples (up to 100 samples) in order to emphasize the quality of the samples retained. Furthermore, the samples must consist of topologies much smaller (up to 10 nodes per topology) than the ones in the test set, to simulate that it is impractical to have production-scale training data, as depicted in Fig. 2. All links must be bidirectional, with a link bandwidth ranging from 10 000 to 400 000 bits and divisible by 1000. Each node may be assigned one of four scheduling policies, and its buffer size must lie between 8000 and 64000 bits. Each pair of source-destination nodes can have only one routing path, although it may differ depending on the direction. We generate traffic flows between node pairs, with an average bandwidth between 10 and 10000 bps, following a time distribution, of which we have three choices, and a size distribution, where we can assign a probability for any size from 256 to 2000 bits, as long as all probabilities sum to 1. Each node pair is associated with a single traffic flow, although node pairs A-B and B-A can have different flow values. Each traffic flow may also be assigned one of three possible types of service, which factors into interactions with node scheduling policies. A comprehensive list of all the constraints that the training dataset must satisfy can be found in [9].

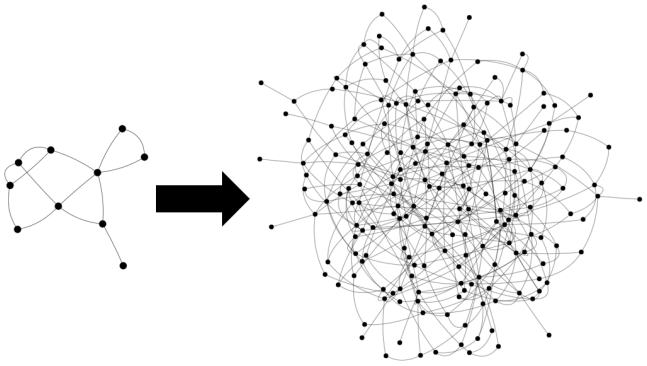

Fig. 2 – Network size: Train with a small network (<10 nodes), Validate/Test with a large network (<300 nodes)

In summary, participants had to generate a training dataset satisfying the aforementioned requirements, and then train the given neural network with this data (where all training hyperparameters are fixed by the challenge organizers). In order to assess the performance of the model after training, which indicates the quality of the generated training data, a dataset containing larger network samples, that we will refer to as the validation dataset in this paper was provided by the competition organizers. At the end of the competition, both the generated dataset (as well as the script to generate it) and the trained model weights must be sent to the competition organizer. They will test the trained model on another dataset, known as the "test" dataset, which contains samples with distributions similar to those of the validation dataset (participants do not have direct access to the test dataset). The team that achieves the lowest MAPE on the test dataset will be considered the winner, provided the organizers are able to reproduce the solution.

3. LITERATURE REVIEW

The problem statement calls for us to synthesize a small dataset for a model that will allow it to generalize to samples on a scale it has not encountered during training. In this section, we review our options for creating datasets, measures available to ensure generalization of the model to domain shift, and whether there are any special considerations when working with very small datasets.

Data generation: Data generation provides an alternative to acquiring real data. This can be done through either crowdsourcing platforms or through simulators [10]. In our case, we must generate our samples through the provided OMNET++ simulator, where generating a large amount of samples is prohibitively costly. Another option for generating data is a generative model, with Generative Adversarial Networks (GANs) in particular being a popular choice. While the constraints prevent us from directly creating new samples using a generative model, we can use one to generate the inputs (topology, routings, traffic matrix) to feed to the simulator.

By inferring the generation parameters of samples in the distribution of data we want to generate from, such as the samples in the validation set, we can train a model to generate samples consisting of generator parameters presumably from the same distribution we will be validating and testing on, which we can then give to the simulator to generate samples. There are some issues with that approach, however. First, generative models require a lot of data for training, without which the discriminator memorizes the training set and fails to generalize, causing the model to fall apart, as [11] shows when a GAN is trained on a small fraction of the CIFAR-10 image dataset (10-20% of 60k images). We can increase the amount of data we have through data augmentation, which is also available for graphs, although the usual augmentation techniques used on images and sequences cannot be applied, as graph structure is not encoded by position [12]. Second, generative models are trained by reconstructing their training examples, which in our case feature topologies much larger than what we can have in our training set. The output shape of a generative model does not have to match the input, as shown by the image degradation and super-resolution tasks [13], but these need ground truth labels, and if we wanted to generate smaller graphs, we would need to train the generative model with a "ground-truth" small graph corresponding to each large graph training example.

Graph model generalization: We next explore what means are available to improve the generalization capabilities of a GNN system. As the competition rules prevent us from altering the model or training procedure, we will focus on means related to data.

Increasing generalizability of GNN models through manipulating the input data generally means performing some form of data augmentation. Data augmentation on graph data is more complex than on image data due to the non-Euclidean nature of the space. Augmentation of graph data can involve the graph's structure, as in [12], which uses an edge predictor to determine edges to add or remove in order to produce new examples. Otherwise, it can act on the feature matrix, by, for example, masking or perturbing some features before passing the matrix through a node propagation model to produce augmented samples, as in [14]. Both of these approaches can be combined, as well. While augmentation is a viable strategy when the problem has no hard upper limit to the number of samples to use for training, in our case we need to improve the quality of the samples passed to the model, as our number of samples is small.

Data cleaning addresses the quality of the samples given to a model for training. The most common approach to this is through removing samples judged deleterious to performance [15]. This "filtering" consists of removing "out-of-distribution" samples from the training, such that the training and test dataset distributions match, but in our case our training and test sets have topologies of different sizes, which implies we have to work with the fact that they come from different distributions. Otherwise, early stopping [16] has been shown to help models generalize better by stopping training before the model starts fitting the noisy data points. As training length is one of the few parameters of the training procedure we have control over in the context of the competition, we can keep this in mind for our solution.

Training on small datasets: There is steadily more attention being paid to training deep learning models on limited data, a problem encountered in certain fields where data acquisition can be difficult, such as medical imaging. Solutions put forth to solve this issue stem mainly from transfer learning, where a model is trained with data for a different task in the hope that it will aid with the main task, and data augmentation, discussed above [17]. Furthermore, altering the model design can also help with learning with scarce data, as demonstrated by [18] who improve the performance of a CNN model on image recognition tasks by changing the loss function from the commonly-used cross-entropy loss to cosine loss. Smaller models can also outperform the state-of-the-art when both are subjected to small training sets, as shown by [19]. Unfortunately, all of the methods surveyed to deal with limited data entail either changes to model architecture and training procedures, or artificially increasing the number of samples within a small dataset, all of which violate the constraints we have to operate under in the context of the GNN competition.

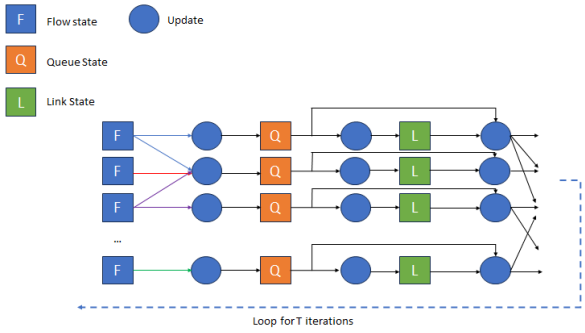

Fig. 3 – Schematic representation of Routenet-Fermi's message passing (adapted from [6])

To our knowledge, there is no research on optimizing the learning of a model with a dataset whose total size is hard-capped at a certain maximum number of samples. References that deal with a limited number of samples usually attempt to overcome this by generating more samples through augmentation. References consulted usually deal with classification problems and define a "small" dataset in terms of the number of examples per class. As the references consulted deal overwhelmingly with classification problems, this leaves regression problems in GNNs as an under-served area of research.

4. NETWORK MODELING

4.1 Background

Network modeling is an important component in the design of networks [6]. Two of the most often used modeling methods are queuing theory and packet-level simulators, which both feature their own limitations. Queuing theory makes strong assumptions about how packets arrive, which may not reflect reality. Packet-level simulators carry a very high computational cost, making them slow and impractical to use in realistic scenarios with many nodes. Modern network modeling techniques can be leveraged to create a "Digital Twin" of the network. A digital twin is a virtual copy of a real-world object (in this case, a network) that can accurately predict how the object would react to a host of simulated situations. Neural networks can be used to create these digital twins [20, 21, 22, 23], and several different architectures can be used for this purpose. The Multilayer Perceptron (MLP), a type of neural network consisting of several fully-connected layers of neurons, can accurately predict traffic generated from a Poisson distribution, but otherwise has trouble anticipating yet-unseen routings [20, 21]. The Recurrent Neural Network (RNN), a model that specializes in sequential data, better supports different traffic models than MLP, but still struggles to cope with routing and topology changes [22, 23]. Graph neural network models

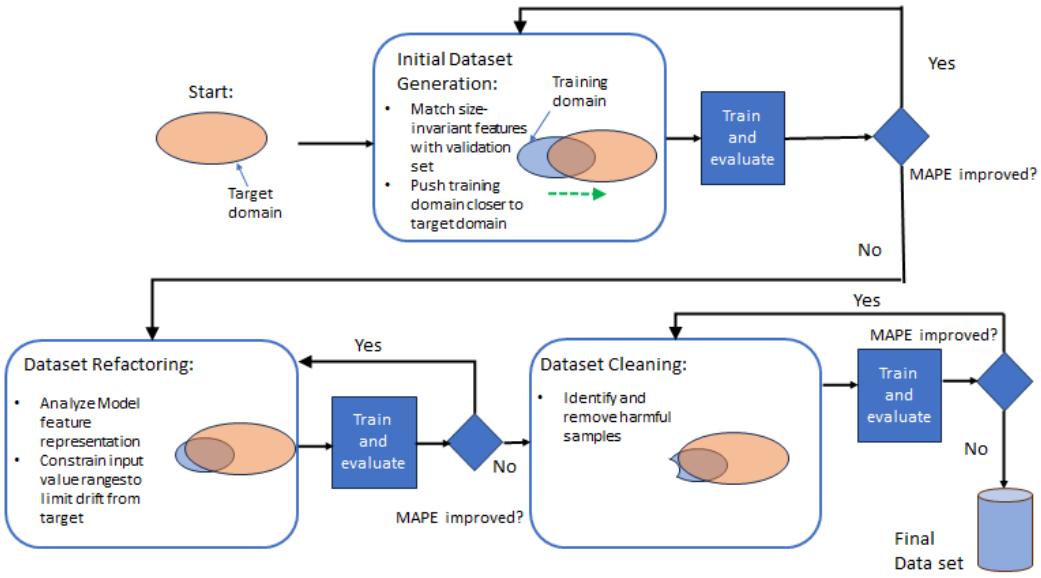

Fig. 4 – Proposed solution overview: A three-step process

directly process graph data as input, but nevertheless still find predicting novel routing configurations difficult, as these configurations are not represented in a simple reproduction of the network topology as graph data.

4.2 RouteNet-Fermi

The RouteNet-Fermi model [6] is a state-of-the-art graph neural network-based model for network simulation. The model goes beyond simply reproducing the network topology as graph data, instead creating a heterogeneous graph that models the interactions between the traffic flows, queues and links of a network. A flow's state depends on the states of links and queues it encounters, a queue's state is influenced by the states of flows that pass through it, and a link's state is influenced by the state of queues that may lead network traffic into it. These circular dependencies are addressed through a three-part, custom message-passing scheme, shown in Fig. 3.

After the states of each of the flows, queues and links are initialized with the initial features, message-passing occurs iteratively, with the queue states being updated first, using aggregated states of the flows encountered by each queue, followed by the link states, using the states of queues that feed into the links, and flow states, using information aggregated from the states of all links and queues that compose a flow. This process repeats for T iterations, where T is a user-defined parameter before the flow states are finally used to compute the performance metric estimations. Furthermore, in a manner similar to the model in [24], RouteNet-Fermi also features design choices such as replacing the numerical "link capacity" value with a relative value representing the traffic load of a link based on its capacity, and the delay is inferred from the queue occupancy rather than predicted directly, which helps retain accuracy even when testing on networks much larger than those experienced during training.

5. PROPOSED SOLUTION

This section describes the process we employed to generate our dataset. The proposed procedure implements a customized three-step solution, shown in Fig. 4. The restrictions on sample topology size means that the training and target domains will have significant differences, so this is partly a domain generalization problem. First, we generated an initial dataset where we match the input variables for the simulator that do not scale with topology with those of the given validation set, pushing the domain of our generated set closer to the target. Then, we refactored the generated dataset limiting the bounds of some input parameters to constrain the training domain so it does not drift too far from the target. And finally, we designed a cleaning framework to keep only high-quality data, discarding samples that hinder performance. These steps are described in detail in the following subsections.

5.1 Initial dataset generation

We started by conducting a detailed analysis of the validation set to allow us to form hypotheses about how specific parameters of the validation sets were generated. For example, we wanted to hypothesize about the assignment of traffic and service type to source and destination node pairs, the average path length, the link capacity values and how these capacities are assigned to the links, the node buffer sizes and how they are assigned, the scheduling policies, etc. Some of the parameters that we observed in the validation set are shown in tables 1 and 2. In these tables, the "values" column shows the set of possible

Table 1 – Hypothesis made on graph nodes and edges after validation set analysis

Parameters	Values	Probabilities
Policies	[FIFO, SP, WFQ, DRR]	[0.25, 0.25, 0.25, 0.25]
Buffer Sizes	[8000, 16000, 32000, 64000]	[0.25, 0.25, 0.25, 0.25]
WFQ weights	[”70,20,10”,”33.3, 33.3, 33.4”,”60,30,10”,”80,10,10”,”65,25,10”]	[0.2, 0.2, 0.2, 0.2, 0.2]
DRR weights	[”60,30,10”,”70,25,5”,”33.3,33.3,33.4”,”50,40,10”,”90,5,5”]	[0.2, 0.2, 0.2, 0.2, 0.2]
Link capacity	[10000, 25000, 40000, 100000, 250000, 400000, 1000000]	based on number of nodes

Table 2 – Hypothesis made on flows after validation set analysis

Parameters	Values	Probabilities
Traffic flow	Uniform(0,1)×I, I∈[1000,2000,3000,4000]	[0.25, 0.25, 0.25, 0.25]
Packet Size Distribution	”0,500,0.22,750,0.05,1000,0.06,1250,0.62,1500,0.05”	0.2
	”0,500,0.08,750,0.16,1000,0.35,1250,0.21,1500,0.2”	0.2
	”0,500,0.53,750,0.16,1000,0.07,1250,0.1,1500,0.14”	0.2
	”0,500,0.1,750,0.16,1000,0.036,1250,0.24,1500,0.14”	0.2
	”0,500,0.05,750,0.28,1000,0.25,1250,0.27,1500,0.15”	0.2
Time Distribution	”Poisson”,”CBR”,”ON-OFF (5,5)”	[1/3, 1/3, 1/3]
Type of Service	0,1,2	[0.1, 0.3, 0.6]

values a parameter can take. The ”probabilities” column contains the probability of selecting a given value. For example, the parameter ”scheduling policy”, abbreviated to ”policies”, can be selected from the set of values [FIFO, SP, WFQ, DRR], each policy can be selected with equal probability, represented by [0.25, 0.25, 0.25, 0.25]. The other parameters in the tables can be interpreted in the same way, with values selected at random based on probabilities.

From the assumptions made based on tables 1 and 2, we can see that apart from link capacity whose values change as the graph size increases, the other parameters seem to be chosen randomly with probabilities defined as in the tables. We made the assumption that we should use these values as input parameters for the simulator in order to generate our training set as well. So we generated a training set by choosing the different parameters as defined in tables 1 and 2. After training the model with the datasets generated according to the process defined above, the best MAPE we could obtain on the validation set was around 27-29%.

5.2 Refactoring the dataset

We were unable to get satisfactory performance when trying to match the validation set's parameters with networks that had no more than 10 nodes. We concluded that our dataset was insufficiently diverse, i.e., that the parameters were not sufficiently variable to account for all potential use cases encountered during testing. We thus looked at other characteristics that could have an impact on the parameters defined in Section 5.1, such as delay ranges, queue utilisation, link utilisation, and average port occupancy.

5.2.1 Link capacity based on routing

Our exploration of the validation set showed that the average link bandwidth in a given topology generally increased proportionally with the number of nodes. We assume this is done to accommodate the increasing number of flows in larger topologies, as the validation follows the same rules as the training set, in that each source-destination pair of nodes has a single flow assigned to it. As we can only have one flow per node pair in topologies of our sample, and with the restriction that the topologies in the training set must not exceed 10 nodes, this implies that the total number of flows will always be much lower than the total found in the validation set. The link utilization analysis we performed on the generated datasets from Section 5.1 showed that our link utilization was very low on average (about 40%). In order to adjust the link utilization so that it covers a wide range, we set the link capacity based on the routing, as this gives us greater flexibility, as explained below. In this strategy, the routing policy is known in advance and we set the capacity of the links to be proportional to the traffic it encounters. We can therefore set the capacity so that we get a desired level of link utilization. Equation *(1)* shows how we obtained the link capacities.

$$\text{capacity}_\ell = \frac{\sum\limits_{f \in N_\ell} t_f}{LU_\ell}, \qquad (1)$$

where t_f is the traffic of the flow f, N_l is the number of flows traversing link l, and LU_ℓ is the link utilization of link l which was chosen from a normal distribution of mean μ and standard deviation $\sigma = \frac{\mu}{2}$. We have chosen the values of the mean μ such that the links experience low, medium and high utilization levels, which is similar to what we have observed in the validation set. Algo-

rithm 1 details the process for obtaining the link utilization LU_ℓ. Notice that the function "random(.,.)" chooses a value from vector "mean" with probability "weights". Choosing LU_ℓ this way adds variability to the dataset, so each example yields a more meaningful contribution.

Algorithm 1: get_load()

```
mean = [0.2, 0.4, 0.6, 0.8]
weights = [0.3, 0.3, 0.3, 0.1]
while True do
    μ = random(mean, weights)
    σ = μ/2
    LU_ℓ = N(μ, σ)
    if 0 ≤ LU_ℓ ≤ 1 then
        break
Return: LU_ℓ
```

Using this approach, the link capacity values concentrated around 10000, which is minuscule compared to link capacities in the validation set. Looking at how the RouteNet_Fermi model [6] estimates the delay of a flow as follows,

$$\text{delay}_f = \sum_{(q,\ell)\in f} \left(\frac{\text{GNN\_}qo_\ell}{capacity_\ell} + \frac{\mu\_fps_f}{capacity_\ell} \right), \qquad (2)$$

we see that the delay is highly dependent on the link capacity, as it is used to calculate the queuing and transmission delay. Since the GNN readout output ($\text{GNN\_}qo_\ell$) may give similar values in the training and validation set, having a link capacity value in the training set that is very different from those in the validation set is counterproductive. To address this, instead of using the link capacity directly derived from *(1)* (hereinafter referred to as 'cap'), we set the link capacity to be the closest candidate to 'cap' from the set {10000, 25000, 40000, 100000, 250000, 400000}, which is a subset of the link capacity in the validation set. Algorithm 2 details the process for setting the link capacities in the network according to the traffic information.

Algorithm 2: set_link_bandwidth()

```
Input: G, paths, traffic, capacity_set
link_bw = Array of 0
foreach pair (SRC, DST) ∈ G do
    path = paths(SRC,DST)
    foreach e in path do
        link_bw(e(SRC), e(DST)) += traffic(SRC,DST)
foreach e in G.edges do
    link_load = get_load()
    cap = link_bw (e(SRC), e(DST)) / link_load
    Id_cap = arg min(|capacity_set − cap|)
    G(e[SRC], e[DST]) = capacity_set[Id_cap]
Return: G
```

5.2.2 Network topology choice

To get the most flows per sample, we decided to only generate 10-node graphs, the maximum permitted. We found that having one unique network topology per sample, so 100 different network topologies in total, was not beneficial, as it could add too much noise to the dataset (Achieved MAPE $\approx$ 16% on the validation set with 100 different network topologies) . Therefore, we carefully designed and selected 10 network topologies to have two important characteristics found in the validation set: the presence of nodes of degree 1 and different node degree configurations, as shown in Fig. 5 and Table 3, to increase the probability of having a different path length distribution using the shortest path algorithm. Indeed, on a 10-node graph, the presence of higher-degree nodes, e.g., 6, limits the path length using the shortest-path algorithm to around 3. Since we wanted to have different path length distributions using the shortest path algorithm, generating graphs on the basis of node degree gave us greater flexibility. Each topology was used to generate 10 samples where only the node and edge characteristics were modified. The node attributes were chosen as described in Table 1 and the edge characteristics were chosen according to Section 5.2.1.

5.2.3 Flows generation

Based on our observations of the validation set, we identified four traffic flow intensities: 1000, 2000, 3000, and 4000 bps, to use as a base. For each sample, we randomly chose an intensity and the flows in that sample were selected from the interval [intensity/2, intensity]. We also found that defining only 10 distinct traffic matrices delivers better performance than 100 different traffic matrices when working with 10 network topologies of 10 nodes each. Thus, we defined 10 different traffic matrices and used these same traffic matrices for each of the different network topologies. In addition, updating the probability distribution values of the "packet size distribution" and "time distribution" parameters we defined in Section 5.1 from [0. 2, 0.2, 0.2, 0.2, 0.2] to [0.1, 0.2, 0.3, 0.3, 0.1] and from [1/3, 1/3, 1/3] to [0.8, 0.1, 0.1] respectively, resulted in flows that produced better results. These new values were found after several search cycles of our algorithm, as shown in Fig. 4. We observe improvement when we generate more flows with a Poisson distribution (probability = 0.8) when we have few samples. In order to easily visualize the similarities between the training and the validation dataset in terms of average flow value we plot Fig. 6, in which we can see that after all the parameter tuning used to generate the training dataset, both violin plots look alike, demonstrating the similarity between the training and validation sets in this particular aspect.

Training the model with the dataset generated after the different changes described in Sections 5.2.1, 5.2.2 and 5.2.3 resulted in a MAPE reduction from 27-28% to 8-10% on the validation set.

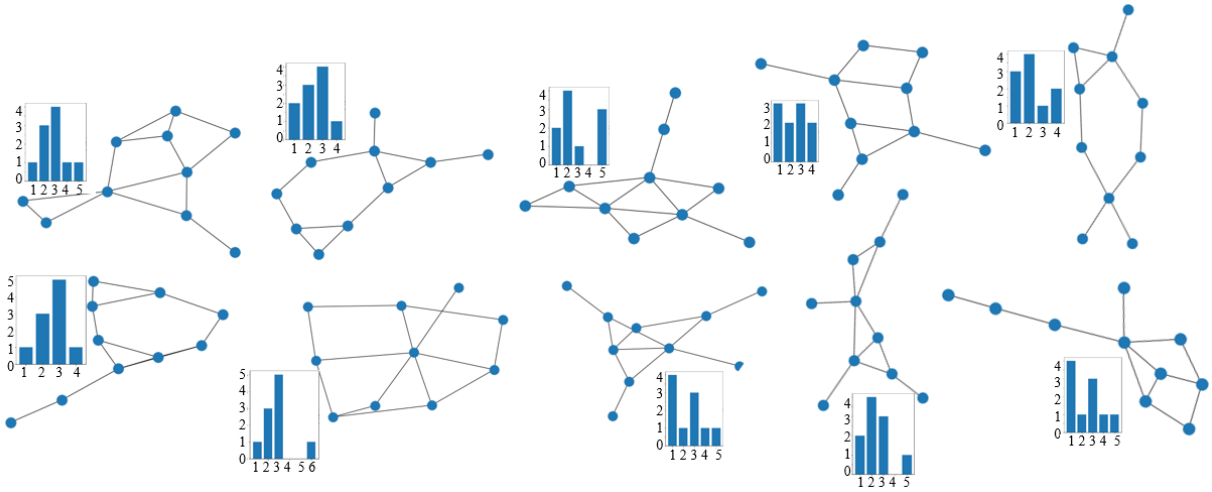

Fig. 5 – Selected network topologies with corresponding degree histogram (degree on x-axis, count on y-axis).

Table 3 – Key topology characteristics.

Topology ID	1	2	3	4	5	6	7	8	9	10
$\|N\|$	10	10	10	10	10	10	10	10	10	10
$\|L\|$	14	12	14	12	11	13	14	12	12	12
Diameter	4	4	4	4	5	5	3	4	5	5
Avg. degree	2.8	2.4	2.8	2.4	2.2	2.6	2.8	2.4	2.4	2.4
Max. degree	5	4	5	4	4	4	6	5	5	5
Min. degree	1	1	1	1	1	1	1	1	1	1
Avg. betweenness	0.12	0.15	0.13	0.15	0.17	0.16	0.11	0.14	0.16	0.16
Max. betweenness	0.47	0.38	0.47	0.38	0.47	0.42	0.52	0.52	0.68	0.64
Min. betweenness	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

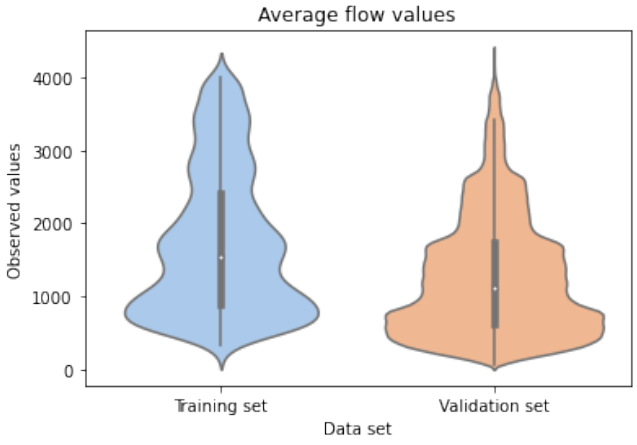

Fig. 6 – Traffic flow distribution Training vs. Validation set.

5.3 Dataset cleaning

With 100 samples generated, we explored whether cleaning this dataset by removing some samples judged too noisy can help improve the results. As what was "Noisy" was difficult to specify, we hypothesized on what can be considered noise.

5.3.1 Hypothesis 1: Noise based on data distribution

Our first hypothesis is: A sample is considered noisy if it contains certain characteristics (e.g., average port occupancy in our case) that are outside the validation set's distribution. We suppose this based on our knowledge of the RouteNet-Fermi model, which predicts the average port occupancy, then derives its delay prediction. Thus, we wanted the average port occupancy distribution of our generated training samples to match that of the validation set. When we cleaned our dataset following this hypothesis, we saw no improvement, and could not confirm our first hypothesis.

5.3.2 Hypothesis 2: Noise based on path length

As we were using the shortest path routing algorithm and that message passing iterated eight times, in order to reduce the over-smoothing during the training, we made a second hypothesis. We suppose that a sample is noisy if it does not contain at least one path of length 4, roughly the average path length in the validation set. We identified 10 samples and found that all these samples were generated with the topology containing a degree 6 node. After these samples were removed, we trained the neural net-

work and the MAPE dropped to 6-7% over the entire validation set, using only 90 samples. We thus demonstrated how we use our understanding of the model to optimize our dataset.

The pseudo-codes describing how the network topologies, the traffic matrices and the final dataset have been generated are given in Algorithm 3, Algorithm 4 and Algorithm 5, respectively.

Algorithm 3: generate_topology()

```
Input: num_nodes, prob, policies,
       buffer_sizes,wfq_weights, drr_weights
G = G_{n,p}(num_ nodes, prob) [25]
for node in G do
    node_schedulingPolicy = random(policies)
    node_bufferSizes = random(buffer_sizes)
    if node_schedulingPolicy == WFQ then
        wfqWeight = random(wfq_weights)
    if node_schedulingPolicy == DRR then
        drrWeights = random(drr_weights)
```

Algorithm 4: generate_traffic()

```
Input: G, intensity, time_dists,td_weights,
       size_dists, sd_weights, tos_list, tos_weights
traffic = matrix of zeros of size G
foreach (SRC,DST) pair ∈ G do
    b_avg = random([intensity/2, intensity])
    td = random(time dists, td weights)
    sd = random(size dists, sd weights)
    tos = random(tos list, tos weights)
    traffic[SRC, DST] = b_avg
Return: traffic
```

Algorithm 5: generate_dataset()

```
Input: num_nodes, prob, capacity_set, policies,
       buffer_sizes, wfq_weights, drr_weights,
       time_dists, td_weights, size_dists,
       sd_weights, tos_list, tos_weights,
       intensity_set
for i ∈ {1,2,...,10} do
    for j ∈ {1,2,...,10} do
        G = generate topology()
        paths = shortest paths routing
        intensity = random(intensity_set)
        traff_ar = generate_traffic()
        set_link_bandwidth()
generate_dataset_simulator() #use omnet++
clean_dataset() # as described in Section 5.3
```

6. NUMERICAL RESULTS

This section presents the results obtained by our methodology, compares the best solutions (in the challenge ranking) to our proposed solution, and highlights our findings.

The experiments have been done on a machine running a Windows system equipped with an AMD processor of 32-core, 2.95 Ghz with 128 GB of RAM and an NVIDIA GeForce RTX 2080 Ti. The code of our solution is available on GitHub[1].

6.1 Results analysis

The improvements obtained with our pipeline at each stage are shown in Table 4. We present the MAPE obtained on the entire validation set referred to as ("ALL"), as well as the MAPE obtained in each subset of the validation set grouped according to the number of nodes in their network topology ("50 nodes", "75 nodes", etc.). As described in Section 5, we can see that we moved from a MAPE of about 28% to around 6% for the entire validation set. We identified some datasets (e.g., those with 170, 200 and 240 nodes) as difficult, with a higher MAPE at each stage. We note that the dataset with the lowest MAPE at each stage (260 nodes) has a low number of samples (9) compared to 14 in other sets. As a baseline, the competition organizers stated that with thousands of samples of networks up to 10 nodes, they were able to get a MAPE of about 5% on the validation set. With our 90 generated samples, we reached a MAPE of around 6%. This demonstrates that it is possible to efficiently train a neural network with smaller datasets if the data is of good quality.

6.2 Comparison against other solutions

When comparing our solution to other finalists at the end of the challenge, a common characteristic seen in all top solutions was the detailed analysis of the validation set to extra insights and generate the initial dataset. Table 5 presents the results obtained by the top three teams of the competition on the test set (our solution ranked first).

***Team Ghost Ducks*[2]:** The second-ranked team generated around 270K samples and trained two Oracle models with about 85K samples from the generated samples. They extracted a vector representation (embedding) for each sample in the validation and training samples. This vector representation contains the path state, link state and length of each flow path. They then clustered the validation set and assigned each training sample to a cluster. Finally, they took the top k samples from each cluster to arrive at a set of 100 samples to get their final traning set. They obtained a MAPE of 8.554 % on the test set.

Team Net: The third-ranked team proposed a beta distribution-based leave-one-out sample ranking strategy. They built their initial dataset by generating different distributions from the beta distribution. They then assign each sample a score indicating the quality of the sample by examining the impact of removing that sample

[1] https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-002-SNOWYOWL-GNNetworking-Challenge2022

[2] https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-002-GhostDucks-GNNetworking-Challenge2022

Table 4 – MAPE (%) obtained on validation sets

Updates	50 nodes	75 nodes	100 nodes	130 nodes	170 nodes	200 nodes	240 nodes	260 nodes	280 nodes	300 nodes	ALL
Section 5.1	27.89	26.66	29.31	32.83	34.43	34.06	31.78	17.32	28.29	25.23	28.26
Section 5.2	7.66	8.19	9.01	10.93	10.94	12.34	10.91	3.85	7.45	6.97	8.57
Section 5.3	6.00	6.22	6.81	7.97	8.47	8.48	8.02	3.16	6.10	5.40	6.50

Table 5 – Top solutions comparison on the test set

	Snowyowl	Ghost Ducks	Net
# samples	90	100	100
MAPE (%)	8.55334	8.55446	9.97016

from the set. They then select the top 100 samples to use as their training set. They obtained a MAPE of 9.79 % on the test set.

The uniqueness of our solution lies in the fact that we did not generate thousands of samples before reducing to a smaller set. Instead, we focused on understanding the data and the model to refactor and clean the 100 generated initial samples. We achieved our best result with only 90 samples, further lending credence to the fact that data quality is more important than volume.

6.3 Our findings

This challenge was an excellent opportunity to explore data-centric AI in the networking domain to formulate a methodology to generate high-quality datasets. From our experiments, we can formulate the following points we believe can help the community create better datasets.

1. As shown when we derived link congestion level and capacity, examining derived statistics for clues to guide setting the bounds on input feature space (here, limiting the range of 'link capacity') helps in elaborating a training set that improves the model's generalization.
2. Attempting to improve model generalization through maximizing the range of possible values for all input features in the training set does not guarantee performance improvement even when our training size is drastically reduced, as we've shown when we achieved our best results by limiting the variety of topologies and traffic matrices in our final training set.
3. The final results justify turning towards data to improve model performance, as we were able to show that the model can retain good performance even when training on a drastically reduced training set(amount of samples reduced to less than 10% of original configuration), as shown by our result of 8.55% MAPE on the final test set, compared to about 5% according to the organizers when training on thousands of samples. This result was achieved without changing the model's architecture, hyperparameters or training procedure.

7. CONCLUSION

This work presents a method to identify and generate relevant training samples, in order to reduce the cost of generating datasets in networking. We proposed a three-step approach, consisting of identifying topology size-independent characteristics to reproduce in our proposed dataset, deriving appropriate topology size-dependent characteristics, and identifying deleterious samples for removal, resulting in a high-quality dataset that, even with only 90 samples of 10-node networks, we can train a model that scales effectively to samples of large networks with 50 to 300 nodes.

Future directions include investigating the quality of the samples through angles not explored in our work. The "cleaning" step used here operates along the dataset topology, seeking to identify samples with detrimental properties in the structure of the nodes and links. We must pay more attention to the traffic flows, and which properties in the flows that form the samples may improve model performance. We require a more robust metric for data quality evaluation to accurately quantify the contribution of each sample.

ACKNOWLEDGEMENTS

This work was supported by two joint Mitacs-Ciena internships (Large-scale optimization for optical and fiber networks & Self-Organized Fabric - SOF projects).

REFERENCES

[1] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio. "RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN". In: *IEEE Journal on Selected Areas in Communications (JSAC)* 38.10 (Oct. 2020), pp. 2260–2270.

[2] Eliza Strickland. "Andrew Ng, AI Minimalist: The Machine-Learning Pioneer Says Small is the New Big". In: *IEEE Spectrum* 59.4 (2022), pp. 22–50. DOI: 10.1109/MSPEC.2022.9754503.

[3] Andrew Ng, and LandingAI team Data-Centric AI. https://landing.ai/data-centric-ai/.

[4] Andrew Ng, Dillon Laird, and Lynn He Data-Centric AI Competition 2021. https://https-deeplearning-ai.github.io/data-centric-comp/.

[5] BNN-UPC Graph Neural Networking challenge 2022. https://bnn.upc.edu/challenge/gnnet2022/.

[6] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "RouteNet-Fermi: Network Modeling With Graph Neural Networks". In: *IEEE/ACM Transactions on Networking* (2023), pp. 1–.

[7] Lester James Miranda. "Towards data-centric machine learning: a short review". In: *github* (2021).

[8] DataNet API Documentation. https://github.com/BNN-UPC/datanetAPI/tree/challenge2021.

[9] BNN-UPC Constraints for the training dataset. https://github.com/BNN-UPC/GNNetworkingChallenge/blob/2022_DataCentricAI/training_dataset_constraints.md.

[10] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. "Data collection and quality challenges in deep learning: A data-centric ai perspective". In: *The VLDB Journal* (2023), pp. 1–23.

[11] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. "Differentiable augmentation for data-efficient gan training". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7559–7570.

[12] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. "Data augmentation for graph neural networks". In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 35. 12. 2021, pp. 11015–11023.

[13] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. "Photo-realistic single image super-resolution using a generative adversarial network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

[14] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. "Graph random neural networks for semi-supervised learning on graphs". In: *Advances in neural information processing systems* 33 (2020), pp. 22092–22103.

[15] Ihab F Ilyas and Theodoros Rekatsinas. "Machine Learning and Data Cleaning: Which Serves the Other?" In: *ACM Journal of Data and Information Quality (JDIQ)* 14.3 (2022), pp. 1–11.

[16] Mingchen Li, Mahdi Soltanolkotabi, and Samet Oymak. "Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks". In: *International conference on artificial intelligence and statistics*. PMLR. 2020, pp. 4313–4324.

[17] Ms Aayushi Bansal, Dr Rewa Sharma, and Dr Mamta Kathuria. "A systematic review on data scarcity problem in deep learning: solution and applications". In: *ACM Computing Surveys (CSUR)* 54.10s (2022), pp. 1–29.

[18] Bjorn Barz and Joachim Denzler. "Deep learning on small datasets without pre-training using cosine loss". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1371–1380.

[19] Lorenzo Brigato and Luca Iocchi. "A close look at deep learning with small data". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 2490–2497.

[20] Nargess Sadeghzadeh, Ahmad Afshar, and Mohammad Bagher Menhaj. "An MLP neural network for time delay prediction in networked control systems". In: *2008 Chinese Control and Decision Conference*. IEEE. 2008, pp. 5314–5318.

[21] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. "Machine learning for networking: Workflow, advances and opportunities". In: *Ieee Network* 32.2 (2017), pp. 92–99.

[22] Salem Belhaj and Moncef Tagina. "Modeling and Prediction of the Internet End-to-end Delay using Recurrent Neural Networks." In: *J. Networks* 4.6 (2009), pp. 528–535.

[23] AysŞe Rumeysa Mohammed, Shady A Mohammed, and Shervin Shirmohammadi. "Machine learning and deep learning based traffic classification and prediction in software defined networking". In: *IEEE International Symposium on Measurements & Networking (M&N)*. 2019, pp. 1–6.

[24] Junior Momo Ziazet, Charles Boudreau, Brigitte Jaumard, and Huy Duong. "Addressing RouteNet scalability through input and output design". In: *ITU Journal on Future and Evolving Technologies* (2022).

[25] Vladimir Batagelj and Ulrik Brandes. "Efficient generation of large random networks". In: *Physical Review E* 71.3 (2005), p. 036113.

AUTHORS

Junior Momo Ziazet received his M.Sc. degree in telecommunications engineering from the National Polytechnic School of the University of Douala, Cameroon in 2017. In 2020, he received another M.Sc. in industrial mathematics from the African Institute for Mathematical Sciences. He is currently pursuing a Ph.D. in computer science at Concordia University, Canada. He was awarded the best paper award at the IEEE International Symposium on Measurements & Networking in 2022. His main research interests focus on large-scale optimization and the application of machine learning and deep learning approaches to communications and network systems.

Charles Boudreau received a B. Sc. and M. Sc. in computer science from Concordia University, Canada in 2018 and 2021, respectively, where he is currently pursuing a PhD. His research interests include graph neural networks, and deep learning applications towards optimization of service function chains in cloud networks.

Oscar Delgado received an M.A.Sc. degree in electrical engineering from Concordia University, Montreal, QC, Canada, in 2010, and a Ph.D. degree in electrical engineering from McGill University, Montreal, QC, Canada, in 2016. He is currently a research associate with the Department of Systems Engineering, École de Technologie Supérieure, Montreal, QC, Canada, where he is the research coordinator of the Energy Efficiency project in partnership with Ericsson GAIA. His current research interests include AI applications for 5G wireless mobile communication technologies, including, network virtualization, and green wireless systems. His focus is on the analysis and design of traffic management techniques, service assurance, resource allocation strategies, and energy efficiency algorithms.

Brigitte Jaumard is a professor in the Computer Science and Software Engineering (CSE) Department at Concordia University. Her research focuses on mathematical modeling and algorithm design (large-scale optimization and machine learning) for problems arising in communication, transportation and logistics networks. She is also a senior advisor for the Montreal Ericsson GAIA (Global Artificial Intelligence Accelerator) research center and the chief scientist of CRIM. Brigitte Jaumard was ranked among the top 2% of scientists in her field of research according to a 2021 study based on research citations. She was awarded several research chairs (Canada Research Chair and Concordia Research Chair, both Tier I during the years 2000-2019). B. Jaumard has published over 300 papers in international journals in operations research and in telecommunications.