Learning for Accelerated Traffic Engineering with Differentiable Network Modeling

Wenlong Ding, *Student Member, IEEE,* Libin Liu, *Member, IEEE,*
Li Chen, *Member, IEEE,* Hong Xu, *Senior Member, IEEE*

***Abstract*—Traffic engineering (TE) in the wide-area networks (WAN) is crucial for optimizing network performance by distributing traffic across various paths. Traditional methods use optimization like linear programs (LP) to solve TE, resulting in long decision-making time especially in larger topologies due to the iterative nature of optimization algorithms. Deep neural networks (DNNs) have been recently applied to TE to expedite decision-making through model inference. However, existing works rely on discrete event-based network simulators or non-differentiable algorithms to compute TE metrics from decisions by interacting with the network environment, breaking the gradient chains from decisions to metrics. Thus, TE metrics are treated as scalar values without gradients, limiting the learning paradigm for TE to reinforcement learning (RL) only.**

This paper demonstrates that calculating TE metrics can be fully differentiable, enabling direct gradient-based DNN updates for better TE decision-making, surpassing RL's reliance on approximated value functions. We propose dNE, a lightweight network simulator that uses differentiable matrix operations to evaluate TE decisions and compute user-defined metrics, enabling advanced DNN training paradigms like goal-driven optimization supervised by TE metric gradients. With dNE, experiments on four DNN-based TE algorithms show that DNNs trained with metric gradients reduce performance loss by over 10x compared to RL, speed up LP solvers by 13000x, and achieve 1000x faster metric computation than traditional event-based simulators.

***Index Terms*—Traffic engineering, deep learning algorithms, differentiable network modeling.**

I. Introduction

Cloud providers use wide-area networks (WANs) to interconnect global data centers and deliver planet-scale applications [25–27]. These WANs are privately owned by certain production companies and centrally controlled following the software-defined networking paradigm referred to as SD-WAN. SDWAN simplifies network management by abstracting complex underlying network infrastructure, which provides high-level network information like traffic demand between source-destination pairs (i.e. flows) and enables centralized control operations such as path assignment and demand-level flow management, instead of traditional configurations that focus on packet-level queuing and forwarding [25–27, 29]. Traffic engineering (TE) is a well-studied topic in SDWAN that aims to optimize a global performance objective such as minimizing the maximum link utilization by allocating traffic demands of flows to their candidate paths with constraints such as link capacity and demand satisfaction [25, 26, 28, 29, 49].

Wenlong Ding and Hong Xu are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong SAR, China. Libin Liu and Li Chen are with the Zhongguancun Laboratory, China.

Modern TE systems need frequent decision updates of traffic allocation (usually every 5 minutes [29]) to adapt to traffic dynamics, and also require fast speed of each update based on real-time traffic to minimize the use of outdated decisions that may result in performance degradation [11, 48]. Traditional TE uses traffic demands and path information obtained from centralized controller in SDWAN to formulate linear programs (LP) to optimize TE objectives [25, 26]. However, both existing works [11, 48] and our empirical evidence in §II-A indicate that runtime for solving LP is non-negligible, and it takes several seconds for a topology with dozens of nodes, which contradicts fast decision-making requirement. Also, when scaling to larger topologies with hundreds or thousands of nodes, its runtime can stretch to several hours, making it further unacceptable for TE systems. Our goal is to accelerate TE decision-making while maintaining good performance on TE objectives similar to conventional LP.

Inspired by the success of reinforcement learning (RL) with deep neural networks (DNNs) in complex online decision-making tasks [40], deep RL (DRL) algorithms have been introduced for TE [21, 42, 44, 49]. Through model inference of well-trained RL agents, they can provide TE decisions timely. Stampa et al. [42] and Xu et al. [49] both adopt the model-free actor-critic based DRL algorithms to solve TE in SDWAN, determining routing paths and traffic demand allocated to these paths. These works rely on simulated network environments, such as OMNet++ [45] and ns3 [24] to obtain RL rewards related to TE objectives for given TE decisions. Recent work Teal [48] integrates GNN with RL agents for fine-grained flow allocation and relies on non-differentiable algorithms and formulations to compute TE metrics as RL rewards.

However, discrete event-based simulators or algorithms have notably slow speeds to simulate the networks and evaluate TE decisions, which aggravates the model training time for DRL-based TE. Evaluation in §VI-D shows that training DRL-based TE takes over 3 days for only 100 epochs with ns3 [24] on a 73-node topology. Recent studies propose DNN-based network simulators [39, 50, 52] to accelerate it. Training these DNNs, however, still needs discrete event-based simulators or algorithms to generate enough training data for different settings and topologies [50, 52], which is also time-consuming.

More importantly, discrete event-based simulators or algorithms are inherently non-differentiable due to their focus on the packet level or finer-grained flows, which exhibits irregular behaviors and cannot be fully modeled mathematically [24, 45, 48]. Thus, evaluated TE metrics are obtained as scalar values rather than differentiable functions derived

from TE decisions. This prevents using promising training paradigms that leverage direct gradients from TE metric functions to guide model updates for improved TE decisions. Consequently, DNN-based TE methods are limited to DRL, which only requires scalar metrics as RL rewards without gradient computation. Essentially, DRL agents strive to approximate non-linear value functions with random sampling for model update, which introduces inherent instability and divergence during training [14, 35], leading to longer training time and less competitive results for TE. Large DRL models also require significant tuning efforts to improve model robustness [31].

Therefore we ask the inevitable question for learning-based TE: Why do we have to stick with DRL and discrete event-based simulators or algorithms? DRL approaches with these traditional simulators or algorithms tend to conduct TE at packet or finer-grained flow level and share the same underlying assumption that computation of TE metrics from decisions in networks cannot be explicitly and differentially modeled. While it is valid in some fine-grained traffic scheduling systems [16, 34], which focus on situations that are stochastic (e.g., low-level flows within datacenter networks [16]) or uncontrollable (e.g., video streaming over Internet [34]), it is unnecessary to focus on low-level behaviors in the context of SDWAN, where most modern TE systems are deployed [25–27]. The logical centralized controller [25, 25, 26] in SDWAN clearly provides high-level information about network, such as demand values of traffic flows, link capacities, and available paths between nodes. Additionally, the controller can automatically distribute TE decisions at demand level, i.e., allocating traffic demand values to each candidate path, rather than addressing irregular behaviors of packets, which are already managed by controller's internal components at backend. This indicates that SDWAN TE process has the potential to be fully-differentiable modeled with careful designs, allowing us to explore various training paradigms beyond DRL.

We introduce dNE, a fully-differentiable network simulator that computes TE metrics from decisions. dNE takes a set of matrices as input that describe network environments obtained from the SDWAN controller, such as traffic demands, routing paths, and link capacities, as well as TE decisions provided by DNN models. It then computes resulting network states and user-defined TE metrics, such as maximum link utilization, using (differentiable) matrix operations. dNE's primary goal is to facilitate exploration of various DNNs and training paradigms for potential performance gains while accelerating TE decision-making. Additionally, dNE with linear algebra is clearly much faster than discrete event-based simulation.

We highlight our contributions as follows.

- We design dNE, a lightweight and fully differentiable network simulator that enables explorations of diverse DNNs and training paradigms for learning-based TE in SDWAN. dNE uses differentiable matrix operations to model the network and evaluate TE decisions, allowing direct gradients of TE metric functions to guide DNN training, overcoming the reliance on DRL training only.
- We prototype dNE[1] using PyTorch's automatic differentiation engine [37], integrating it as a differentiable "plugin" that can be directly embedded into any DNN training framework. We also provide functions and templates for user-defined TE metrics, network modeling and flexible use of dNE's modules. To demonstrate its usability, we implement three goal-driven DNN-based TE algorithms trained with metric gradients and a traditional DRL algorithm using dNE.
- Using dNE, evaluations show that our new goal-driven DNN-based algorithms can reduce the optimality gap by up to 12.6x than the existing DRL-based TE algorithm and deliver 13374x faster decision-making time than LP. Moreover, dNE provides 1037x simulation acceleration during model training when compared to traditional simulator ns3 [24]. Additionally, dNE's robust performance on extreme topologies with thousands of nodes further demonstrates its strong scalability.

[1]https://github.com/NetX-lab/dNE.

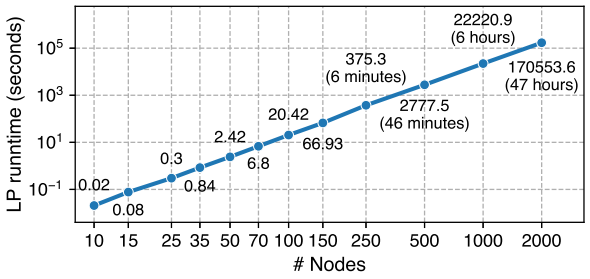


Fig. 1: LP solving time for TE in different topologies with random edges that are 3x the node count. For topologies with less than 500 nodes, results are averaged over 100 runs on the hardware detailed in §VI; for those with more nodes, results are averaged over 5 runs on the same machine with an extra 768G memory to run Gurobi solver. Both x- and y-axis are in log scale.

II. Background & Motivation

We begin with the background of SDWAN TE, then discuss existing DRL algorithms and their limitations, and finally motivate dNE.

A. TE in SDWAN

TE in SDWAN allocates traffic of each source-destination pair (i.e. a flow) to its available paths considering various constraints like link capacities, aiming to optimize a TE objective such as maximum link utilization (MLU) [25, 26, 28, 29, 49]. To deal with traffic dynamics, modern TE systems necessitate frequent updates (e.g. every 5 minutes [11, 29]) to traffic allocation decisions according to real-time traffic demand of the network. During the interval between updates, quick decision-making is crucial to minimize the time spent with outdated decisions that could potentially lead to performance degradation during each update cycle. The centralized controller in SDWAN provides a white box network environment for TE, including traffic demand, available routing paths for each flow and link capacity, which are sufficient for formulating TE objectives such as MLU [25, 26, 29]. In addition, SDWAN controller applies TE decisions to network at traffic demand level which only considers demand value on paths and is irrespective of low-level packet behaviors [25, 26].

Following the characterization by Kumar et al. [29], TE system makes two specific decisions for each flow in its formulation: (1) candidate paths, which can be multiple possible paths from the source node to its destination of current flow, and (2) splitting ratios, referring to the fraction of traffic demand allocated to each candidate path for current flow. For candidate paths, production SDWANs typically use a predetermined path set and remain this set unchanged during online TE [11, 25, 26, 29, 49]. For example, Smore [29] uses oblivious routing [13] to select k candidate paths for each flow, while other works [11, 49] simply use k-shortest paths. Splitting ratios are usually determined online periodically such as every 5 minutes mentioned above. A traditional approach to obtain optimal splitting ratios is to formulate linear programs (LP) to optimize the TE objective [25, 26, 29]. Other works, known as equal-cost multi-path routing (ECMP), equally distribute traffic across candidate paths [15, 18, 20]. ECMP eliminates the time required to solve optimization problems in trade of TE performance, as demonstrated in our evaluation in §VI-B. TE decisions in our work will follow the manner mentioned above, which calculates splitting ratios online periodically based on pre-determined candidate paths.

While LP provides optimal performance, its runtime is non-negligible. As shown in Fig. 1, a 70-node topology requires over 5 seconds for TE decision-making. Moreover, LP runtime escalates rapidly with the increase in topology size, leading to extremely huge decision-making time for larger topologies. Fig. 1 shows that the runtime is 0.02 seconds for a 10-node topology, but it extends to about 2 days for that of 2000 nodes. The non-negligible runtime and substantial time increase in larger topologies are undesirable for TE, which cause TE system to experience performance loss with outdated decisions during LP solving, or it may not even finish solving new decisions within the entire update period (e.g. 5 minutes). Thus, we are tempted to use DNNs, which make TE decisions through model inference, potentially bringing faster decision speeds and slower runtime increases.

B. DRL for TE

Along with the success in solving complex online control problems [40], DRL algorithms are used to deal with TE in WAN [21, 33, 42–44, 49], including recent work Teal [48]. These algorithms work to determine how to distribute traffic to candidate paths for each flow to achieve optimization objectives. By leveraging the model inference of RL agents, they offer fast and efficient TE decision-making in response to fluctuating network traffic. However, these approaches rely on discrete event-based network simulators or algorithms, such as OMNet++ [45], ns3 [24] and non-differentiable formulations [48], to simulate network environments and evaluate TE decisions, and they basically distribute traffic at packet or finer-grained flow level. This is because these TE works focus on these lower-level behaviors, such as packet queuing and forwarding, which behave irregularly and cannot be fully modeled mathematically.

However, relying on discrete event-based network simulators or algorithms lead to inefficient TE. First, discrete event-based network simulators or algorithms have slow speed to evaluate TE decisions, which limits training speed. Empirical evidence in §VI-D shows that training DRL for TE on a topology with 73 nodes for only 100 epochs should take over 70 hours with the commonly used traditional simulator ns3 [24]. Second, it limits DNN-based TE approaches only to DRL algorithms. These simulators have inherent non-differential properties because the lower-level behaviors are hard to model, which cannot support more promising training paradigms that are supervised by direct gradients from TE metric functions. Such paradigms require a differentiable function mapping TE decisions to metrics rather than treating TE metrics as scalar values without gradients as existing methods. Those DRL-based algorithms may lead to less optimistic TE performance due to the inherent limitations of DRL, e.g. sampling DRL actions inefficiently [14] and large variance of training results between two consecutive training epochs [35]. Third, per-packet traffic distribution among selected paths is not practical in reality. The low-level information (such as packet-level) is hard to obtain in a timely manner. For example, traffic collection tools in production, e.g., iFIT [3] and sFlow [7], can only collect low-level data every five minutes, because network devices cannot afford the computational resources to implement more accurate and timely collection.

In SDWAN, information is abstracted at the flow-demand level rather than the packet level, with demands for all flows clearly provided by the centralized controller [25–27]. The controller also assigns flows to certain candidate paths based on specific demand values solved by TE algorithms, while the controller backend can handle packets to ensure the entire network remains aligned with the decisions. This avoids solving irregular low-level behaviors with DNN models which existing works struggle with, paving the way for improved DNN-based TE approaches[2].

C. dNE's Benefits

dNE's primary goal is to create a network simulator that has a fully-differentiable process for evaluating TE decisions and computing TE metrics by interacting with network environment. It should enable direct gradients from TE decisions to metrics, allowing direct DNN model updates through how the metrics correspond to the decisions. Unlike traditional methods relying on discrete event-based simulators or algorithms that compute TE metrics as scalar values without gradients, dNE eliminates the need for value function approximation or random sampling in DRL, resulting in faster convergence and better performance. So the first advantage of dNE is its differentiable computation, which enables direct gradient guidance and allows the use of goal-driven optimization [17, 41] supervised by gradients of TE metric functions to train DNNs instead of DRL's approximate value functions. The second advantage is that dNE is designed as a plugin easily integrable into any DNN training framework, facilitating the exploration of different DNN architectures for various use cases.

To achieve this, dNE leverages the principles of *differentiable programming* [19, 30], where programs are designed to

[2]We specifically focus on SDWAN in this paper, so methods are not suitable for fine-grained control systems [16, 34] that focus on packet behaviors.

derive gradients automatically using Automatic Differentiation (AD) [37, 46, 47]. For example, PyTorch [6] uses built-in differentiation engine, `torch.autograd` [37], to construct a computational graph that traces operations from input tensors to output tensors, enabling gradient computation via the chain rule. dNE implements its fully differentiable SDWAN simulator following this paradigm, supporting gradient propagation from TE decision to metrics and finally to learning models. By modeling the network environment and evaluating TE decisions as differentiable matrix operations, dNE eliminates time-consuming simulations at low-level, allowing faster computation of TE metrics.

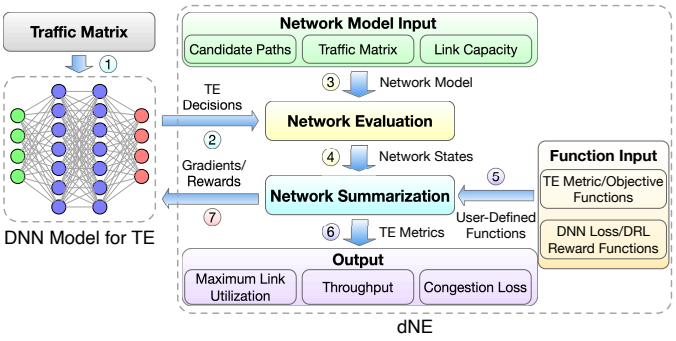


Fig. 2: dNE overview.

III. Design

We present dNE's designs in this section. We first illustrate the overall workflow of dNE in §III-A, and then describe its network model in §III-B. Lastly, we introduce dNE's key components in §III-C.

A. System Overview

dNE is a lightweight network simulator that evaluates TE decisions and outputs corresponding TE metrics using fully-differentiable functions that interact with the network environment in SDWAN. Its simulation operations are modeled as a series of differentiable matrix operations as described in §III-C. Designed as a plugin, dNE can be seamlessly integrated into the training frameworks of various deep learning models and learning paradigms, including goal-dirven optimization with direct metric gradients unexplored before, where TE metric-related functions are directly used as the loss function for gradient updates. It is also applicable to traditional DRL, where the TE metric is treated as a scalar.

Overall Workflow. dNE has two stages to analyze and evaluate TE decisions: Network Evaluation and Network Summarization. Evaluation stage calculates the statistics of network states after applying given TE decisions from DNNs, providing necessary information for defining various performance metrics. Summarization stage focuses on processing the user-defined functions and outputting the corresponding TE metrics, gradients or rewards. Former stage only involves internal calculations within dNE, while the latter is open for users to program. Note that dNE's key design principle of calculations is to ensure entire process is differentiable and can be integrated into gradient chains, rather than simply calculating a TE metric as a scalar as done in previous works. Details of dNE's overall workflow is shown in Fig. 2.

1) First, DNN model outputs TE decisions for all flows according to current traffic, and dNE takes them as input. ① When new traffic matrix arrives, the DNN receives it as input. ② DNN performs inference and outputs corresponding TE decisions for evaluation. As discussed in §II-A, we focus on online performance on pre-determined paths, so decisions here refer to splitting ratios as most of related works [25, 29, 49].
2) Second, Network Evaluation stage analyzes TE decisions to obtain related network states. ③ It first evaluates decisions in network environment, where necessary network components, such as candidate paths, traffic, and link capacity, are designed to be modeled as matrices. ④ Then, it outputs the evaluated states, which users can directly utilize to define their functions. For instance, the aggregated traffic demand on each link can be derived from the evaluated state and used to compute common TE metrics, such as MLU.
3) Third, Network Summarization stage calculates user-defined TE metrics. ⑤ Users can utilize network states and network models to define and program their own TE metric functions, which are then stored in dNE. ⑥ Initially, dNE provides differentiable formulations of common TE metrics, such as MLU, throughput, and congestion loss, as mentioned in [29].
4) Finally, DNN model is updated with metric gradients. ⑦ Gradient chain in Summarization stage is calculated during the process of computing TE metrics with user-defined functions, which are then used to update DNN model. Alternatively, the gradient can also be ignored, and the TE metric can be treated as a scalar reward to develop a DRL algorithm with dNE. For example, (user-defined) MLU function can serve as a training loss function or its negative value can be used as a DRL reward to serve both gradient-based and DRL training.

In this workflow, DNNs only take the traffic matrix as input, while topology information, such as link capacity and candidate paths, is implicitly incorporated into the dNE plugin during training. This design ensures that DNNs focus exclusively on traffic features without being influenced by the static topology, leading to more accurate TE decisions and avoiding duplication information in model inputs. But please note that DNNs in our framework do not include any (static) neurons that simulate or encode the network topology. They are solely responsible for producing TE allocation results based on traffic input, while the network context is evaluated by dNE. dNE with these topology information exists only within the training framework instead of the models themselves.

B. Network Model

We model necessary network environment information for evaluating TE decisions, including candidate paths, traffic, and link capacity in dNE's current design for example as shown in Fig 2. These can be modeled as matrices or vectors.

	Notation	Definition
Inputs	C_e	Capacity on link e
	D_f	Traffic demand on flow f
	P_e^i	Binary encode indicating whether link e is on path i
Auxiliary	L_e	Aggregated traffic demand on link e
TE decision	W_i^f	Traffic fraction of flow f on path i (Splitting ratios)

TABLE I: Notations used in dNE.

Consider the topology as a graph $G = (V, E)$, where V represents set of nodes and E the set of links. As shown in Table I and Fig. 3, link capacity in topology can be modeled to vector C with the length of $|E|$. Traffic demand is usually in the form of matrices in production SDWAN [28, 29, 36], and we flatten it into a vector D to facilitate the following matrix operations of dNE shown in Fig. 3. We have $|F| = |V| \times |V|$ flows, so vector D is with the length of $|F|$. For each flow, there are k candidate paths used to distribute its traffic, so we have $k \times |F|$ paths in total. The binary path-link relationship matrix P is with the size of $(|E|, k \times |F|)$, where P_e^i equals 1 if path i traverses through link e and otherwise 0. After obtaining splitting ratios for all flows in DNN model, we transform them to flow-path relationship matrix W with the size of $(k \times |F|, |F|)$. We fill W_i^f with 0 if path i is not the candidate path for flow f, otherwise the relative splitting ratio obtained from DNN will be assigned. As outlined in Smore [29], modern SDWAN controllers require dNE's TE decisions to allocate all demands for each flow, ensuring the splitting ratios across its k candidate paths sum to 1. This can be achieved by applying *softmax* [8] function in the output layer of a DNN model.

C. Network Evaluation and Summarization

In this section, we show detailed designs of Network Evaluation and Network Summarization components in dNE.

Network Evaluation. Evaluation stage plays a crucial role in providing users with a clear understanding of network's states after applying given TE decisions to current network environment. Network states should be modeled to indicate real-time network statistics, which are unambiguously determined and accessible to users to calculate custom metrics. In dNE's current design, we use aggregated traffic on each link as our example network states, which is enough to support common TE metrics calculation mentioned in Smore [29] as we will show in Summarization stage. The aggregated traffic can be represented as a vector L with length $|E|$. The vector format enables easy matrix operations within dNE.

L_e should sum up all splitting traffic from all flows which are distributed to link e. As Fig. 3 shows, based on candidate paths and traffic demand (matrix P and D) in network model and the given TE decisions (matrix W), we calculate L_e using

$$L_e = \sum_f \sum_i P_e^i W_i^f D_f. \tag{1}$$

In modern SDWAN [29], TE decisions do not constrain aggregated traffic on links within capacity ($L_e \leq C_e$) but instead allocate all traffic demands as discussed before, discarding excess traffic per link. A common TE metric to measure this demand loss is congestion loss [29], as discussed later.

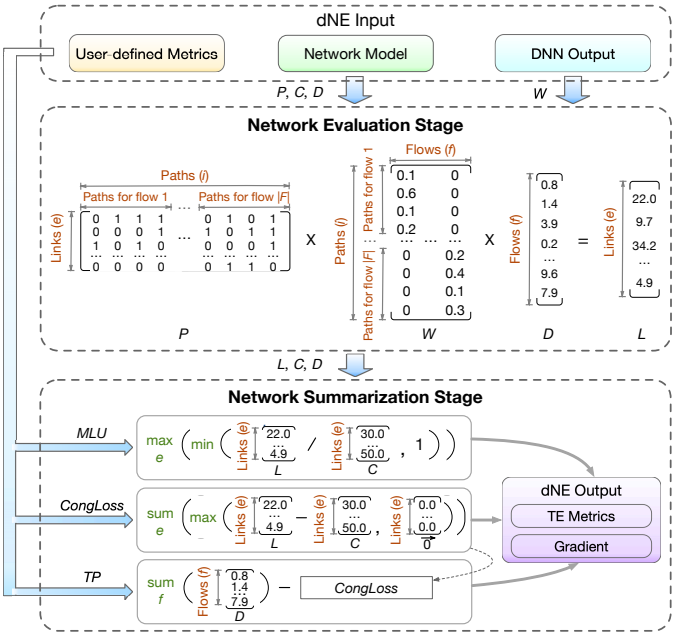


Fig. 3: Matrix operations for network evaluation and summarization.

Network Summarization. Network Summarization summarizes real-time network performance with various user-defined TE metrics and computes DNN loss/reward functions based on the network states obtained from the Evaluation component. The Summarization stage is customizable and can be programmed by users using available variables in network states and models. Details can be found in §V.

dNE initially defines three common and crucial TE metrics [29]: maximum link utilization (MLU), congestion loss ($CongLoss$), and throughput (TP). They are also widely used in most of famous TE works [11, 25, 26, 28, 29, 49]. They can be calculated by a series of simple matrix operations such as maximum, minimum, and summation, as shown in Fig. 3. The details are as follows:

- *Maximum link utilization (MLU).* This represents the maximum value of link utilization among all links. Note that a link cannot carry traffic that is more than its capacity. So link utilization value is up to 1, and traffic exceeding its capacity will be dropped. That is,

$$MLU = \max_e(\min(L_e/C_e, 1)). \tag{2}$$

- *Congestion loss.* We calculate the dropped traffic due to congestion on all links. Since we satisfy all traffic demands when considering TE decisions, a link may carry traffic more than its capacity, and exceeding parts will be dropped. We sum up all dropped traffic on all links across the topology. That is,

$$CongLoss = \sum_e \max(L_e - C_e, 0). \tag{3}$$

- *Throughput.* We calculate actual transmitted demand after dropping the congested traffic. That is,

$$TP = \sum_{f} D_f - CongLoss. \tag{4}$$

The optimization objectives of DNNs can be TE metrics described above. Users can also design more complex loss functions for gradient-based learning methods or reward functions for DRL that are related to the specific objective TE metric. Using AD, dNE is able to calculate the gradients and update the models under the guidance of these functions.

IV. DNN-based TE Use Cases

We introduce several simple examples to show how to design DNN algorithms with dNE in order to solve TE problems. Three typical DNNs will be introduced: fully connected neural network (FC) [38], long short-term memory (LSTM) [22], and convolutional neural network (CNN) [23]. DRL method following previous work [49] will also be implemented in its dNE version. For all DNN models, the original input is the unflattened traffic matrix with the size of $|V| \times |V|$, and it outputs TE decisions (splitting ratios) with the size of $k \times |V| \times |V|$. Then the output values will be used to form matrix W for dNE.

Our TE objective here is to minimize *MLU*, a widely used metric in TE research [25, 29, 49, 51], which balances traffic load across network links. We use goal-driven optimization [17, 41] with dNE, which directly utilizes the *MLU* calculation function as the loss function to update DNNs. We avoid the traditional supervised paradigm, as it requires computing ground-truth TE metrics (scalar labels) for different sample traffic and topologies using linear programming, which is highly time-consuming (§II-A). For DRL-based approaches that maximize rewards, dNE uses the negative value of the computed *MLU* as a scalar reward to minimize *MLU*.

FC-TE. In FC-TE, input layer flattens original traffic matrix, and output layer provides splitting ratios in vector form directly. Besides, we use several hidden layers in FC. Neurons in these layers use activation functions (e.g., *ReLU* [12]) to add non-linear factors, enhancing the expression ability of the model. FC is the most basic and simplest DNN that we can come up with. It has a simple neural network structure and a relevant small model size, making it easy to train with little inference time. But at the cost of these advantages, it may not always give optimistic performance because its simple structure may not adequately express the complexity of the problems we need to solve.

LSTM-TE. We input one flow's demand into one LSTM cell sequentially, thus entire model has $|V| \times |V|$ cells in total. Then, we concatenate hidden states in all cells as input of a fully connected neural layer, which outputs the splitting ratios. Assuming that each LSTM cell H hidden states, the fully connected layer has $H \times |V| \times |V|$ input neurons. We regard the model as allocating traffic flow by flow: each cell's hidden layer represents current flow's allocation information, and latter flows can consider decisions of previous flows by passing information in sequential cells. With a unified loss function, each flow adjusts its own hidden states to make an overall good performance for all flows.

This structure can explore each flow's strategy in individual cell while considering relationship between nearby input flows, possibly leading to better model expression ability and performance than FC-TE. However, LSTM-TE may not perform well in large topologies because: 1) With too many LSTM cells, later ones can hardly remember previous information due to LSTM's inherent short-term feature; 2) It takes much inference time in long LSTM sequence.

CNN-TE. In CNN-TE, we maintain traffic input in its original 2D-matrix form (i.e., length and width are both $|V|$, and channel size is 1). For each layer in learning model, besides convolution operations, we apply batch normalization to enable fast and stable training, add *ReLU* activation to add non-linear factors, and apply max pooling to compress the data/parameters and avoid over-fitting. Then, the output of the final CNN layer will be fed into a fully connected neural layer, which will output the splitting ratios.

Obviously, CNN-TE can benefit from spatial features in original traffic matrix, leading to possible better performance. Also, it may alleviate LSTM-TE's problems: 1) CNN-TE can consider the relationship between flows without forgotten features in LSTM; 2) As topology scale increases, model inference in CNN-TE needs one-pass of the same structured model just with a larger input width, but LSTM-TE needs to pass more LSTM cells and its number has rapid growth speed, leading to more TE decision time. Evaluations in §VI have proven our arguments here. But certainly, CNN-TE is harder to train and has more inference time than FC-TE since it has more complex model structure.

DRL-TE. Its design is similar to previous work [49], but the key difference is that we rely on dNE for reward calculation instead of traditional simulator. And the RL states are real-time network dynamic environments, including traffic matrix, aggregated traffic on links and throughput. The RL actions provide splitting ratios of all flows and its reward is the negative value of MLU. We use the actor-critic structure to implement RL [49], where the actor network receives RL states as a vector and outputs RL actions, critic network uses the concatenated vector of RL states and real-time RL actions as input and outputs a scalar representing the reward expectation. As we have mentioned in §II, RL has its own limitations such as sample inefficiency and variance in results, which may result in long training time to converge with less optimistic TE results. We implement DRL-TE in dNE's version just as a comparison to the newly proposed DNNs above trained with metric gradients.

Users can choose the most appropriate DNN-based approach based on specific needs following these use cases.

V. Implementation

We implement dNE with fully-differentiable computing and network model building. The detailed code is provided in the footnote following the corresponding contribution item in §I. Specifically, we are able to prototype all matrix operations in our design (§III) using PyTorch [6], leveraging its automatic differentiation (AD) support [37].

We utilize these operations to implement dNE's two key stages: Evaluation() and Summarization(), and provide template and example files demonstrating how to use these functions to create TE algorithms with goal-driven optimizations and metric gradients for any DNN model. Users can also register custom TE metric functions in Summarization().

Additionally, we build utilities to transform topology data, including candidate paths, traffic demands, and link capacities, into the matrix format required by dNE. These inputs can be sourced from XML files in the Internet Zoo Topology dataset [4] which can then be tackled with NetworkX package in Python [5], or from plain text formats in famous TE simulation tool Yates [28].

VI. Evaluation

We are mainly interested in answering the following three questions in dNE's evaluations:

- How do various DNN-based TE algorithms perform? Can we find better ones beyond DRL-TE with dNE? (§VI-B)
- Can DNNs trained with dNE accelerate TE decision-making? (§VI-C)
- Can dNE accelerate DNN-based TE training? (§VI-D)

Environment Settings. We have two computing environments: a GPU environment with an Nvidia GeForce RTX 3090 GPU, and a CPU environment with 4 Intel Xeon Platinum 8268 24-Core processors and 128GB RAM. We use the GPU one for model training and inference of DNN-based algorithms, while the CPU one is used for traditional CPU-only TE methods, such as LP, and other evaluations, including DNN training time evaluation on traditional simulators. Except for GPU one, model inference time will also be evaluated in CPU environment to ensure a fair comparison with LP optimization, which is only conducted on CPU. For TE setups, we use $k = 4$ candidate paths per flow calculated with oblivious routing method mentioned in Smore [29].

DNN Settings. Our DNN model settings are as follows: FC-TE has three hidden layers with 256, 512, and 256 neurons in each layer, respectively. LSTM-TE has a hidden state dimension of 16 for each LSTM cell. CNN-TE consists of three layers with output channels number of 16, 32, and 64 in each layer. It also uses a 3×3 kernel size with a stride length of 1 for convolution, and a 2×2 kernel size with a stride length of 1 for max pooling. DRL-TE uses two hidden layers, with 512 and 256 neurons in each layer, for both the actor and critic networks that are FC DNN models.

Training Details. We apply common training techniques to gain best performance of all DNN models. We reduce *learning rate* by a factor of 10 every 20 epochs, achieving larger initial rates for faster convergence and smaller rates later for finer optimization. Training stops when loss change over 5 epochs is below 10^{-6} (convergence) or after 100 epochs. We test starting learning rates of 0.1, 0.01, and 0.001 and report the best results. We *normalize* DNNs' inputs (traffic demand matrices) to a range of 0 to 1 to improve model generalization and stabilize training. We use Adam [1] as *optimizer* of training, which improves over SGD [9] by accelerating convergence with momentum and reducing training oscillations. We also apply weight decay [10] to optimizer, which prevents overfitting by adding regularization terms to loss functions, improving model generalization. We have tested decay values of 0.1, 0.01, and 0.001 and report the best results.

Topology	# Nodes	# Links	# Traffic Matrices	Granularity
Abilene	12	30		5 mins
Geant	22	72	4500 + 500	15 mins
Iris	51	128	(Train + Test)	1 hour
Intellifiber	73	194		1 hour

TABLE II: Evaluated network topologies with bi-directional links.

A. Experiment Setup

Topologies and traffic traces. As shown in Table II, Abilene and Geant are two backbone networks widely used in existing TE works [28, 29, 32], and we obtain their topologies and 5000 real traffic matrices from SNDLib [36], with a granularity of 5 and 15 minutes for Abilene and Geant, respectively. Iris and Intellifiber are two real topologies from Internet Topology Zoo [4], with 5000 realistic traffic matrices each generated using YATES [28] with a granularity of 1 hour. Note that this section focuses on topologies with dozens of nodes, as these topologies are commonly studied in traditional TE works [25, 26, 29, 49], while more extreme topologies are evaluated in §VII to show the good scalability of dNE. We use the first 4500 traffic matrices for model training and the rest for testing for all topologies. The learning models are trained offline and deployed online for inference only during testing. We scale all traffic following the method in Smore [29] to ensure the minimum MLU of testing is 0.4, reflecting traffic patterns observed in traditional overprovisioned SDWANs [25, 26]. We set the capacity of all links to 1Gbps for all topologies. All evaluation results are on average of 500 testing matrices.

Schemes compared. Besides the DNN-based TE algorithms training with dNE mentioned in §IV, we also compare two traditional methods:

- *ECMP*: Traffic of a flow should be evenly distributed to its candidate paths as previous work [20]. Specifically, since we have $k = 4$ candidate paths for a flow, each candidate path should carry 25% of its traffic.
- *Optimal*: LP problem is formulated to obtain the optimal TE decisions, which is similar to prior work [25, 29]. Recall that our TE objective is minimizing MLU (§IV), following notations in Table I, we should compute the splitting ratios $\{W_i^f\}$ with LP formulated as follows:

$$\min \ Z \tag{5}$$

$$\text{s.t.} \sum_i W_i^f = 1, \forall f, \ \text{path } i \text{ belongs to flow } f. \tag{6}$$

$$W_i^f \geq 0, \forall i, f, \ \text{path } i \text{ belongs to flow } f. \tag{7}$$

$$L_e = \frac{\sum_f \sum_i P_e^i W_i^f D_f}{C_e}, \forall e. \tag{8}$$

$$L_e \leq Z, \forall e. \tag{9}$$

Constraints (6) and (7) enforce that TE decisions satisfy demands for all flows and all traffic allocations are non-

Topology	Optimal	ECMP	FC-TE	LSTM-TE	CNN-TE	DRL-TE
Abilene	0.6649	0.9735	0.7183	0.7019	0.6837	0.9447
Geant	0.5461	0.9392	0.6167	0.5992	0.5891	0.9244
Iris	0.6370	0.7411	0.6602	0.6829	0.6405	0.7292
Intellifiber	0.6452	0.8034	0.6890	0.7114	0.6478	0.7772

TABLE III: Average maximum link utilization (MLU).

Topology	Optimal	ECMP	FC-TE	LSTM-TE	CNN-TE	DRL-TE
Abilene	1.0	0.9372	0.9999	1.0	1.0	0.9330
Geant	1.0	0.9835	1.0	1.0	1.0	0.9776
Iris	1.0	0.9992	1.0	1.0	1.0	0.9996
Intellifiber	1.0	0.9970	1.0	0.9995	1.0	0.9982

TABLE IV: Average normalized throughput.

Topology	Optimal	ECMP	FC-TE	LSTM-TE	CNN-TE	DRL-TE
Abilene	0.0	0.0628	0.0001	0.0	0.0	0.0670
Geant	0.0	0.0165	0.0	0.0	0.0	0.0223
Iris	0.0	0.0008	0.0	0.0	0.0	0.0004
Intellifiber	0.0	0.0030	0.0	0.0005	0.0	0.0018

TABLE V: Average normalized congestion loss.

Topology	*MLU*			*Normalized throughput*			*Normalized congestion loss*		
	Optimal	ECMP	CNN-TE	Optimal	ECMP	CNN-TE	Optimal	ECMP	CNN-TE
Abilene	0.6799	0.9819	0.6912	1.0	0.9381	1.0	0.0	0.0619	0.0
Geant	0.5732	0.9411	0.5989	1.0	0.9846	1.0	0.0	0.0154	0.0
Iris	0.6298	0.7713	0.6314	1.0	0.9995	1.0	0.0	0.0005	0.0
Intellifiber	0.6501	0.8211	0.6514	1.0	0.9977	1.0	0.0	0.0023	0.0

TABLE VI: TE metrics in training set.

negative. Constraints (8) and (9) limit link utilization to its maximum value Z for all links. We implement this LP in the CPU environment using Gurobi Optimizer [2].

B. Performance on TE Metrics

We show evaluation results of TE metrics defined in §III-C on different methods. All results in this section are evaluated on the testing set using dNE and DNNs trained on the training set unless stated otherwise. We start by analyzing MLU, which is our TE objective. Overall, Table III shows that the new goal-driven optimization methods (FC-TE, LSTM-TE, and CNN-TE) newly proposed in our work always outperform the RL-based method (DRL-TE) and the traditional ECMP method in all topologies, indicating the effectiveness of DNN learning with goal-dirven optimizations in solving the TE problem with dNE's help. Averaged over the four evaluated topologies, ECMP and DRL-TE suffer from 39.81% and 36.57% additional MLU compared to Optimal, but only 7.85%, 8.19% and 2.91% for FC-TE, LSTM-TE and CNN-TE, respectively. Our new methods significantly reduce additional MLU by up to 13.7x and 12.6x compared to ECMP and DRL-TE, respectively. Furthermore, DRL-TE can only reduce MLU by approximately 2.5% compared to traditional ECMP, highlighting the importance of exploring DNN-based TE methods beyond DRL with dNE framework in order to improve its performance on TE metrics.

The MLU results also validate our analysis of the pros and cons of the new methods discussed in §IV. First, CNN-TE always achieves the lowest MLU due to its strong ability to capture spatial features in its 2D matrix input, reducing additional MLU by 2.7x and 2.8x compared to FC-TE and LSTM-TE on average of four topologies. Second, FC-TE performs worse than LSTM-TE in the medium-sized topologies due to its simple FC structure and limited model expression capability. LSTM-TE reduces the additional MLU by 1.4x compared to FC-TE on average of Abilene and Geant. However, in larger topologies, FC-TE outperforms LSTM-TE due to its short-term memory in longer LSTM sequences. On average of Iris and Intellifiber, FC-TE reduces 1.7x additional MLU compared to LSTM-TE.

Results for throughput and congestion loss (Table IV and Table V) under the MLU objective are in line with previous findings on MLU. First, ECMP and DRL-TE perform the worst in all topologies, evidenced by higher normalized congestion loss and suboptimal throughput. On average, ECMP and DRL-TE have normalized congestion loss of 2.1% and 2.3%, respectively, while all other methods have much reduced loss below 0.02%. Second, CNN-TE outperforms other DNN-based methods with the optimal throughput. Third, FC-TE may not be optimal in some topologies such as Abilene, with a 0.01% higher congestion loss compared to Optimal. LSTM-TE may not perform well in larger topologies, with a 0.05% additional congestion loss in Intellifiber.

One may be interested in the same metrics on training set. Table VI shows CNN-TE's results as a representative since it is the most efficient model we have evaluated so far. We compare its performance with optimal LP and the baseline ECMP. Results on the training set show similar conclusions to those on the testing set, indicating that our DNNs are well-trained without issues such as overfitting. On average of four topologies, ECMP incurs 39.34% additional MLU, while CNN-TE incurs only 1.6%. CNN-TE also demonstrates optimal performance in throughput and congestion loss, consistent with its performance on the testing set.

C. TE Decision Acceleration

This section evaluates the runtime (i.e. TE decision time) of various TE algorithms, which refers to LP solving time for Optimal and model inference time for DNN-based algorithms. ECMP employs a static evenly distributed scheme without a decision time. Given the near-to-optimal performance on TE metrics, we explore whether DNN-based TE methods have significant runtime acceleration over Optimal. To ensure a fair comparison, we report model inference time in both CPU and GPU environments, as Optimal can only be executed on CPUs.

Overall, Table VII shows that DNN-based methods exhibit faster TE decision speed compared to Optimal. On average of four topologies, the decision-making in FC-TE, LSTM-TE, CNN-TE and DRL-TE are 13374x, 87x, 1734x and 10196x faster than Optimal in GPU inference, and 1066x, 16x, 99x and 889x faster in CPU inference, respectively. Among DNN-based algorithms, LSTM-TE is the most time-consuming method in both two environments, especially in

Topology	Optimal	ECMP	FC-TE		LSTM-TE		CNN-TE		DRL-TE	
			GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU
Abilene	35.7913	-	0.1077	0.3223	1.2534	5.2685	0.4319	1.2680	0.1318	0.3046
Geant	225.6312	-	0.1184	0.4731	3.0774	25.3436	0.5606	2.1612	0.1463	0.5127
Iris	2201.3738	-	0.1576	1.3708	21.0783	104.6545	0.8974	18.8933	0.1877	1.6551
Intellifiber	6723.6895	-	0.1803	3.2472	47.4246	247.2045	1.6823	46.1550	0.2468	4.0321

TABLE VII: Average TE decision time (milliseconds).

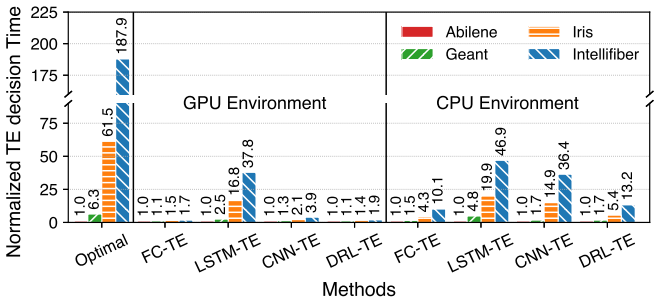


Fig. 4: TE decision time normalized to Abilene for all methods.

large topologies. For example, in GPU environment, LSTM-TE takes approximately 50ms to conduct inference in Intellifiber, which is 263x and 28x slower than FC-TE and CNN-TE, respectively. FC-TE and DRL-TE have similarly fast TE decision speeds due to their simple FC model structures, taking less than 0.3ms and 5ms in GPU and CPU environments, respectively. CNN-TE has a moderate decision time between FC-TE and LSTM-TE in all topologies. These results support our earlier analysis in §IV.

Second, DNN-based algorithms show good scalability for larger topologies, with TE decision times increasing at a slower rate than Optimal in both two environments. This is illustrated in Fig. 4, where the time of all topologies is normalized to Abilene for each method. For example, in Intellifiber. growth rate of DNN-based algorithms ranges from 1.7x to 37.8x in GPU environment and 10.1x to 46.9x in CPU environment, while Optimal is 187.9x compared to Abilene. Fig. 4 also shows that growth rate follows a similar pattern to their absolute decision time, with LSTM-TE growing the fastest, followed by CNN-TE, and DRL-TE and FC-TE the slowest in both environments. Furthermore, GPU inference time increases much faster than CPU inference for all DNN-based methods, highlighting the benefit of GPU inference acceleration for TE over traditional LP that can only process on CPU, especially scaling to larger topologies.

In addition, note that though DRL-TE and ECMP have near-to-zero decision time, they perform poorly on TE metrics compared to new methods, making them not good choices for TE. Alternatively, FC-TE has same TE decision time with better performance, while CNN-TE also has negligible decision time (<2ms) with even better performance.

D. Training Acceleration

We compare dNE with traditional discrete event-based simulators, in terms of accelerating the model training process for TE. We use DRL-TE training as an illustration, which has been widely explored with the traditional simulators in prior works [21, 33, 42, 44, 49]. We compare the commonly used traditional simulator ns3 [24] with dNE in both CPU and GPU environments by training DRL-TE for 100 epochs. Table VIII shows the whole training time and the corresponding network simulation time within dNE and ns3.

Topology	Training time (hours)			Simulation time (hours)		
	dNE (GPU)	dNE (CPU)	ns3	dNE (GPU)	dNE (CPU)	ns3
Abilene	0.567	0.628	0.854	0.013	0.075	0.303
Geant	0.777	0.966	1.909	0.014	0.204	1.151
Iris	1.643	3.321	15.998	0.017	1.688	14.401
Intellifiber	3.111	9.620	73.367	0.022	6.534	70.278

TABLE VIII: The whole training time and simulation time for DRL-TE with dNE and traditional simulator ns3 [24]. Note that simulation time only refers to the process of obtaining RL rewards with TE decisions in the simulator, while training time includes all training parts, such as model forwarding, backpropagation, updates, network simulation, etc.

Overall, evaluation results indicate that dNE significantly accelerates model training compared to ns3, especially in the GPU environment where traditional simulators cannot run. For the entire training time, dNE is 9.32x and 3.94x faster than ns3 in GPU and CPU environments on average of four topologies. In the largest topology, i.e. Intellifiber, it achieves 24x faster training in GPU environment. Additionally, the gain is more evident when only simulation time is considered, which refers to the time from receiving the TE decisions to obtaining the calculated RL rewards. dNE accelerates the simulation part by 1036.8x and 7.24x on average compared to Optimal in GPU and CPU environments, respectively. It can reach up to 3194x faster simulation than ns3 in Intellifiber in GPU environment. Note that dNE only replaces traditional simulators in model training, while other training parts like model forwarding and backpropagation are hard to accelerate, and our high simulation acceleration rate shows dNE's effectiveness in accelerating the entire training process.

Also, dNE exhibits superior scalability compared to ns3 regarding training time as topology size increases. For example, the training time of ns3 in Intellifiber, Iris, and Geant is 85.9x, 18.7x, and 2.2x higher than Abilene, while the values in dNE are only 5.5x, 2.9x, and 1.4x in GPU environment, and 15.3x, 5.3x, and 1.5x in CPU environment, respectively. The same trend holds for simulation time, where the time in Intellifiber is 232x higher than Abilene for ns3 but only 1.7x and 87.1x for dNE in GPU and CPU environments, respectively. Furthermore, the simulation time of ns3 becomes the bottleneck in training with topology scaling, constituting only 35% of whole training time in Abilene but 96% in Intellifiber. However, this value in dNE decreases from 2% to 0.7% in GPU environment. As simulation time in dNE grows much slower than ns3, it takes less fraction of whole training time in larger topologies and avoids network simulation becoming time bottleneck.

One may be curious about the training time of goal-driven optimization paradigm with direct metric gradients, which we present in Table IX. We still use CNN-TE as a representative. The results show that training can be completed within half an hour for all modern topologies, whereas DRL-TE takes more than 3 hours for Intellifiber (Table VIII). This is because CNN-

Topology	Modern topologies				Extreme topologies			
	Abilene	Geant	Iris	Intellifiber	G_{600}	G_{800}	G_{1000}	G_{2000}
Training time (minutes)	4.12	6.78	13.13	20.80	52.08	88.60	132.52	511.81

TABLE IX: DNN training time (minutes) of various topologies with GPU.

Topology	Decision time			Speedup	
	Optimal (LP) (hours)	CNN-TE CPU (ms)	CNN-TE GPU (ms)	CNN-TE (CPU)	CNN-TE (GPU)
G_{600}	1.320	192.67	4.23	2.5x10^4	1.1x10^6
G_{800}	3.315	341.08	6.89	3.5x10^4	1.7x10^6
G_{1000}	6.172	534.41	10.91	4.2x10^4	2.0x10^6
G_{2000}	47.376	2230.31	43.38	7.6x10^4	4.1x10^6

TABLE X: Average TE decision time of 5 runs in the four extreme topologies and the decision speedup of CPU and GPU inference in CNN-TE compared to Optimal. Speedup refers to the ratio of the TE decision time in Optimal to the time of current method.

TE with our new learning paradigm achieves faster learning convergence due to the direct gradient guidance from metric function, unlike the function approximation used by DRL-TE.

VII. Scaling to Extreme Topologies

We have demonstrated in §VI that DNN-based algorithms with dNE exhibit competitive TE performance and fast decision-making speed in topologies with dozens of nodes that are commonly studied in prior classic TE works [25, 26, 29, 49]. However, this prompts questions about its scalability to larger topologies in recent days. Given the network expansion and increasing service demand, recently some global cloud providers have continually added data sites to their networks, causing SDWAN to expand to include hundreds or even thousands of nodes [11, 48]. The primary goal of this section is to evaluate whether DNN-based TE with dNE can maintain its high performance of TE objective and fast decision-making speed in these extreme topologies.

We randomly generate four extreme topologies (G_{600}, G_{800}, G_{1000}, G_{2000}) with 600–2000 nodes and links 3x the node count to evaluate CNN-TE, the representative DNN-based TE method with dNE, which has a good balance between TE decision time and performance as shown in §VI. We generate 5000 traffic matrices and split training and testing set following methods in §VI-A. We use the same hardware stated in §VI, except for increasing server memory to 768GB to run LP in extreme topologies. We also increase the max pooling kernel size of CNN-TE from 2×2 to 16×16 due to limited GPU memory in RTX 3090. Other settings are the same as in §VI. We compare CNN-TE with Optimal and ECMP.

Table XI shows that CNN-TE performs well in all extreme topologies with only 1.8% additional MLU on average compared to Optimal method, while ECMP has much higher additional MLU of 23.0%, which is 13x than that of CNN-TE. In addition, Table X shows that CNN-TE provides significant runtime acceleration compared to Optimal, being $O(10^6)$ and $O(10^4)$ faster in all topologies in GPU and CPU environments, respectively. LP running in Optimal takes hours or even up to 2 days in these topologies, while CNN-TE takes less than 50ms

Topology	Optimal (LP)	CNN-TE	ECMP
G_{600}	0.6320	0.6451	0.7632
G_{800}	0.6154	0.6233	0.7821
G_{1000}	0.6284	0.6397	0.7562
G_{2000}	0.6387	0.6521	0.7913

TABLE XI: MLU of the four extreme topologies.

in GPU environment, achieving timely decision-making. Also, with topology scaling, the speedup ratio of CNN-TE's runtime to Optimal's also increases in both environments, indicating better decision time acceleration for larger topologies and good scalability in CNN-TE. The robust MLU performance and significant decision acceleration of CNN-TE demonstrate that dNE maintains its advantages in these extreme topologies.

Table IX shows that training CNN-TE for all extreme topologies can be efficiently completed within only half a day. Note that traditional linear programming takes over 2 days to solve just a single traffic matrix (Fig. 1). Additionally, this training process occurs offline before TE algorithm is deployed, so this time does not affect online performance.

VIII. Discussion

We first discuss recent related works [11, 48] that also aim to reduce LP runtime in extreme WAN topologies (§VIII-A), followed by the application of dNE in a running TE system for model refinement, where new traffic matrices arrive continuously at each timeslot (§VIII-B).

A. Related Works for Large-Scale TE

NCFlow [11] clusters nearby nodes in topology and runs LP in each cluster in parallel. However, this still takes minutes to solve individual LP, which is far slower than our newly proposed CNN-TE with dNE (<1 second with GPU inference). Moreover, this separate LP approach results in suboptimal performance due to its lack of a global topology view in each LP formulation and the difficulty of developing an accurate merging algorithm for individual cluster results.

Teal [48] uses a complex DNN combining GNNs and DRL to accelerate TE decision-making by capturing network topology and adjusting subflows, achieving fine-grained control for specific operational needs. However, its discrete-event based TE metric calculations, using ADMM for subflow tuning, restrict calculation results to scalars. Thus, Teal still relies on DRL training with slower convergence and suboptimal metrics due to DRL's inefficient value function approximations and action sampling, without leveraging direct metric gradients for model updates. These fine-grained control designs are inefficient for general SDWAN operating at the demand of entire flows (e.g., Smore [29]). In contrast, our method with dNE uses differentiable TE metric functions with its direct gradients, optimizing demand-level TE efficiently. Specifically, we use simpler CNN-TE with dNE than Teal with DRL to achieve near-optimal performance within 50ms on large topologies (∼2000 nodes), compared to Teal's several seconds while still suffering from DRL-induced TE inaccuracy.

Topology	Abilene			Geant			Iris			Intellifiber		
# Iterations	1	5	10	1	5	10	1	5	10	1	5	10
Time (ms)	9.2	40.2	77.3	41.6	193.4	360.2	200.1	923.7	1783.2	479.3	2017.5	3998.61

TABLE XII: Gradient update time (milliseconds) for CNN-TE with GPU environment for a single traffic matrix across different iteration runs.

B. dNE for Running TE System

We have shown that dNE has strong performance and scalability compared to existing methods (§VI and §VII). One may wonder how to better utilize dNE in a running TE system, where single new traffic matrix arrives every timeslot, to further enhance accuracy of learning models. An intuitive approach might be running a few gradient update iterations before making TE decisions. However, Table XII and Table VII show that running just one iteration for CNN-TE takes ∼10x longer than its decision time. Running additional iterations for better accuracy would further increase the time required. Since fast decision-making is critical to avoid outdated TE decisions, which is one of dNE's primary design goals, this significant time cost makes such an approach impractical.

Instead, we propose a similar but more efficient method: for each newly arrived traffic matrix, we use it only for inference during the current timeslot while conducting gradient updates at system backend for future timeslots. This fine-tuning approach improves potential accuracy without delaying TE decision-making. Our evaluation shows that it improves accuracy by 3.1% (CNN-TE) on Abilene compared to approach that does not fine-tune the model after it goes online.

IX. Conclusion

We propose dNE, a lightweight network simulator designed to evaluate the performance of TE decisions in SDWAN. It employs a set of fully differentiable matrix operations to compute TE metrics of given decisions and enables direct gradient chains from metrics to update DNN models. dNE is implemented as a plugin within a training framework, allowing users to train and explore the performance of any DNN model on specific tasks and enabling other promising training paradigms beyond DRL, such as our new goal-driven optimization method with direct metric gradients presented in this paper, for potentially better TE performance. With the help of dNE, we identify several DNNs with our new training paradigm that are better suited for SDWAN TE. Our evaluations demonstrate that they can achieve near-optimal TE performance, significantly outperforming conventional DRL approaches. Furthermore, dNE facilitates fast TE decision-making and model training, offering excellent scalability, especially on larger network topologies.

Acknowledgments

This work is supported in part by funding from the Research Grants Council of Hong Kong (GRF 11209520, CRF C7004-22G) and from CUHK (4937007, 4937008, 5501329, 5501517).

References

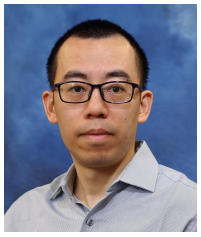
[1] Adam Optimizer. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.
[2] Gurobi Optimizer. http://www.gurobi.com.
[3] Huawei IFIT. https://info.support.huawei.com/info-finder/encyclopedia/en/IFIT.html.
[4] Internet Topology Zoo. http://www.topology-zoo.org/.
[5] NetworkX. https://networkx.org/.
[6] PyTorch. https://pytorch.org/.
[7] SFlow Tool. https://inmon.com/technology/sflowTools.php.
[8] SGD Optimizer. https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html.
[9] SGD Optimizer. https://pytorch.org/docs/stable/generated/torch.optim.SGD.html.
[10] Weight Decay. https://discuss.pytorch.org/t/weight-decay-parameter/83023.
[11] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. Contracting Wide-Area Network Topologies to Solve Flow Problems Quickly. In *Proc. USENIX NSDI*, 2021.
[12] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv preprint arXiv:1803.08375*, 2018.
[13] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal Oblivious Routing in Polynomial Time. In *Proc. ACM STOC*, 2003.
[14] Matthew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement Learning, Fast and Slow. *Trends in Cognitive Sciences*, 23(5):408–422, May 2019.
[15] Jiaxin Cao, Rui Xia, Pengkun Yang, Chuanxiong Guo, Guohan Lu, Lihua Yuan, Yixin Zheng, Haitao Wu, Yongqiang Xiong, and Dave Maltz. Per-Packet Load-Balanced, Low-Latency Routing for Clos-Based Data Center Networks. In *Proc. ACM CoNEXT*, 2013.
[16] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. AuTO: Scaling Deep Reinforcement Learning for Datacenter-Scale Automatic Traffic Optimization. In *Proc. ACM SIGCOMM*, 2018.
[17] Chen, Wenqing and Sim, Melvyn. Goal-Driven Optimization. *Operations Research*, 57(2):342–357, March 2009.
[18] Marco Chiesa, Guy Kindler, and Michael Schapira. Traffic Engineering with Equal-Cost-Multipath: An Algorithmic Perspective. *IEEE/ACM Transactions on Networking*, 25(2):779–792, April 2016.
[19] Jonas Degrave, Michiel Hermans, Joni Dambre, et al. A Differentiable Physics Engine for Deep Learning in Robotics. *Frontiers in neurorobotics*, 13(6):1–9, March 2019.
[20] M Dzida, M Zagozdzon, Michal Pioro, and A Tomaszewski. Optimization of the Shortest-Path Routing with Equal-Cost Multi-Path Load Balancing. In *Proceedings of International Conference on Transparent Optical Networks*, 2006.
[21] Nan Geng, Tian Lan, Vaneet Aggarwal, Yuan Yang, and Mingwei Xu. A Multi-Agent Reinforcement Learning Perspective on Distributed Traffic Engineering. In *Proc. IEEE ICNP*, 2020.
[22] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, October 2000.
[23] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent Advances in Convolutional Neural Networks. *Pattern Recognition*, 77:354–377, May 2018.
[24] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network Simulations with the ns-3 Simulator. *SIGCOMM Demonstration*, 14(14):527, August 2008.
[25] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Rogger Wattenhofer. Achieving High Utilization Using Software-Driven WAN. In *Proc. ACM SIGCOMM*, 2013.
[26] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally Deployed Software Defined WAN. In *Proc. ACM SIGCOMM*, 2013.
[27] Umesh Krishnaswamy, Rachee Singh, Paul Mattes, Paul-Andre C Bissonnette, Nikolaj Bjørner, Zahira Nasrin, Sonal Kothari, Prabhakar Reddy, John Abeln, Srikanth Kandula, Himanshu Raj, Luis Irun-Briz, Jamie Gaudette, and Erica Lan. OneWAN is Better than Two: Unifying a Split WAN Architecture. In *Proc. USENIX NSDI*, 2023.
[28] Praveen Kumar, Chris Yu, Yang Yuan, Nate Foster, Robert Kleinberg, and Robert Soulé. YATES: Rapid Prototyping for Traffic Engineering Systems. In *Proc. ACM SOSR*, 2018.

[29] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-Oblivious Traffic Engineering: The Road Not Taken. In *Proc. USENIX NSDI*, 2018.
[30] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte Carlo Ray Tracing Through Edge Sampling. *ACM Transactions on Graphics*, 37(6):1–11, December 2018.
[31] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*, 2015.
[32] Zhifeng Liu, Zhiliang Wang, Xia Yin, Xingang Shi, Yingya Guo, and Ying Tian. Traffic Matrix Prediction Based on Deep Learning for Dynamic Traffic Engineering. In *Proceedings of IEEE Symposium on Computers and Communications*, 2019.
[33] Tahira Mahboob, Young Rok Jung, and Min Young Chung. Optimized Routing in Software Defined Networks–A Reinforcement Learning Approach. In *Proceedings of International Conference on Ubiquitous Information Management and Communication*, 2019.
[34] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proc. ACM SIGCOMM*, 2017.
[35] Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. Variance Reduction for Reinforcement Learning in Input-Driven Environments. *arXiv preprint arXiv:1807.02264*, 2018.
[36] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. SNDlib 1.0: Survivable Network Design Library. *Networks: An International Journal*, 55(3):276–286, May 2010.
[37] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.
[38] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer Perceptron: Architecture Otimization and Training. *International Journal of Interactive Multimedia and Artificial Intelligence*, 4(1):26–30, September 2016.
[39] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, October 2020.
[40] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, January 2016.
[41] Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, Claire J. Tomlin. Goal-Driven Dynamics Learning via Bayesian Optimization. *arXiv preprint arXiv:1703.09260*, 2017.
[42] Giorgio Stampa, Marta Arias, David Sanchez-Charles, Victor Muntés-Mulero, and Albert Cabellos. A Deep-reinforcement Learning Approach for Software-Defined Networking Routing Optimization. *arXiv preprint arXiv:1709.07080*, 2017.
[43] Penghao Sun, Zehua Guo, Junfei Li, Yang Xu, Julong Lan, and Yuxiang Hu. Enabling Scalable Routing in Software-Defined Networks with Deep Reinforcement Learning on Critical Nodes. *IEEE/ACM Transactions on Networking*, 30(2):629–640, April 2021.
[44] Pravati Swain, Uttam Kamalia, Raj Bhandarkar, and Tejas Modi. CoDRL: Intelligent Packet Routing in SDN Using Convolutional Deep Reinforcement Learning. In *Proceedings of IEEE International Conference on Advanced Networks and Telecommunications Systems*, 2019.
[45] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.
[46] Arun Verma. An Introduction to Automatic Differentiation. *Current Science*, 378(7):804–807, April 2000.
[47] Fei Wang, James Decker, Xilun Wu, Gregory Essertel, and Tiark Rompf. Backpropagation with Callbacks: Foundations for Efficient and Expressive Differentiable Programming. In *Advances in Neural Information Processing Systems*, 2018.
[48] Zhiying Xu, Francis Y Yan, Rachee Singh, Justin T Chiu, Alexander M Rush, and Minlan Yu. Teal: Learning-Accelerated Optimization of Traffic Engineering. *arXiv preprint arXiv:2210.13763*, 2022.
[49] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-Driven Networking: A Deep Reinforcement Learning Based Approach. In *Proc. IEEE INFOCOM*, 2018.
[50] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-Level Visibility. In *Proc. ACM SIGCOMM*, 2022.
[51] Jianwei Zhang, Xinchang Zhang, Meng Sun, and Chunling Yang. Minimizing the Maximum Link Utilization in Multicast Multi-Commodity Flow Networks. *IEEE Communications Letters*, 22(7):1478–1481, July 2018.
[52] Qizhen Zhang, Kelvin KW Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. Mimicnet: Fast Performance Estimates for Data Center Networks with Machine Learning. In *Proc. ACM SIGCOMM*, 2021.

Wenlong Ding is currently pursuing his Ph.D. degree in Department of Computer Science and Engineering, The Chinese University of Hong Kong. He received his B.E. degree with honors in Computer Science and Technology from Huazhong University of Science and Technology, China, in 2021. His current research interests include machine learning for various network management tasks, with a specific focus on network traffic and configuration management tasks.

Libin Liu received his Ph.D. degree from the Department of Computer Science, City University of Hong Kong and his B.E. degree in software engineering from Shandong University. He is currently an Assistant Researcher with the Zhongguancun Laboratory. His current research interests include data analytics systems and machine learning for networking. He received the best paper award of ACM SIGCOMM 2022. He is a member of ACM and IEEE.

Li Chen received the B.E. degree, M.Phil. degree and Ph.D. degree from The Hong Kong University of Science and Technology (HKUST) in 2011, 2013 and 2018, respectively. He previously worked as a Systems Researcher in Huawei Theory Lab. His research interests include Data Center Networks, Distributed Systems, Vert Computing, Machine Learning and OAM. He now works at Zhongguancun Laboratory.



Hong Xu is an Associate Professor in Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research area is computer networking and systems, particularly big data systems and data center networks. From 2013 to 2020 he was with City University of Hong Kong. He received his B.Eng. from The Chinese University of Hong Kong in 2007, and his M.A.Sc. and Ph.D. from University of Toronto in 2009 and 2013, respectively. His work has received best paper awards from ACM SIGCOMM 2022, IEEE ICNP 2023 and 2015, among others. He was the recipient of an Early Career Scheme Grant from the Hong Kong Research Grants Council in 2014. He is a senior member of IEEE and ACM.