Transferable Neural WAN TE for Changing Topologies

Abd AlRhman AlQiam
Purdue University

Yuanjun Yao
Meta

Zhaodong Wang
Meta

Satyajeet Singh Ahuja
Meta

Ying Zhang
Meta

Sanjay G. Rao
Purdue University

Bruno Ribeiro
Purdue University

Mohit Tawarmalani
Purdue University

ABSTRACT

Recently, researchers have proposed ML-driven traffic engineering (TE) schemes where a neural network model is used to produce TE decisions in lieu of conventional optimization solvers. Unfortunately existing ML-based TE schemes are not explicitly designed to be robust to topology changes that may occur due to WAN evolution, failures or planned maintenance. In this paper, we present HARP, a neural model for TE explicitly capable of handling variations in topology including those not observed in training. HARP is designed with two principles in mind: (i) ensure invariances to natural input transformations (e.g., permutations of node ids, tunnel reordering); and (ii) align neural architecture to the optimization model. Evaluations on a multi-week dataset of a large private WAN show HARP achieves an MLU at most 11% higher than optimal over 98% of the time despite encountering significantly different topologies in testing relative to training data. Further, comparisons with state-of-the-art ML-based TE schemes indicate the importance of the mechanisms introduced by HARP to handle topology variability. Finally, when predicted traffic matrices are provided, HARP outperforms classic optimization solvers achieving a median reduction in MLU of 5 to 10% on the true traffic matrix.

CCS CONCEPTS

• **Networks** → **Traffic engineering algorithms**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Traffic Engineering, Wide-Area Networks, Network Optimization, Machine Learning

ACM Reference Format:
Abd AlRhman AlQiam, Yuanjun Yao, Zhaodong Wang, Satyajeet Singh Ahuja, Ying Zhang, Sanjay G. Rao, Bruno Ribeiro, and Mohit Tawarmalani. 2024. Transferable Neural WAN TE for Changing Topologies. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, Sydney, NSW, Australia, 17 pages. https://doi.org/10.1145/3651890.3672237

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672237

1 INTRODUCTION

Cloud and Internet Service Providers (ISPs) must ensure their wide-area networks are managed efficiently to meet the stringent bandwidth and latency requirements of applications. A key component of achieving these goals is to efficiently route traffic to meet desired objectives. Over the last decade, it has increasingly become the norm to use centralized controllers that make traffic engineering (TE) decisions while optimizing network-wide objectives (e.g., minimizing link utilizations, maximizing bandwidth served etc.) [24, 26].

Conventionally, controllers make routing decisions relying on optimization techniques (e.g., linear programming). Unfortunately, as the scale of network topologies increases, conventional optimization methods are not responsive enough given the need to adapt traffic at fine time-scales. Recently, researchers [48, 61] have argued for an alternate approach that involves the use of neural models to solve optimization problems. Not only is inference using neural models much faster than traditional optimization models but also they offer the potential to jointly perform prediction (e.g., of a future traffic matrix) and optimization (of the predicted matrix) [48].

While ML-driven TE offers exciting potential, a central challenge for ML driven approaches is their ability to handle scenarios beyond what they are trained for. This is especially critical in the TE context for two reasons. First, achieving high performance is critical, and the consequences of a poor TE decision can lead to violations of Service Level Objectives (SLOs). Second, WANs continually evolve with the addition of new data-centers, nodes and links, as network architects strive to keep up with the demands on their infrastructure. Further, failures are common in large-scale WANs owing both to planned maintenance events and unplanned failures [17, 18, 40, 53]. Constant changes in topology also imply tunnels (to carry traffic) change over time.

Existing ML-based TE approaches are not explicitly designed for such changes in topology. DOTE [48] uses a neural architecture that assumes the topology is fixed, and does not model changes to nodes, edges, and tunnels, or changes to link capacity. While TEAL [61] does allow for some topology changes, it is designed for a fixed set of tunnels, and is sensitive to the order in which tunnels are supplied as inputs.

In this paper, we present HARP, a first step towards designing ML-based TE scheme that can better handle scenarios outside those that it has been trained for. HARP is designed with two key ideas. First, it is explicitly designed to **ensure transferability through natural invariances** which ensure HARP's outputs are robust to reordering of nodes, the corresponding traffic demands and tunnels. Second, HARP uses an architecture **aligned to the optimization**

models as we believe doing so can better approximate the original (slower) optimization model when encountering unseen conditions. This leverages recent advances in neural algorithmic reasoning that better align architectures and training objectives to obtain more robust out-of-distribution predictions [5, 6].

To achieve these ideals, HARP's architecture has three key components: (i) a Graph Neural Network (GNN) to produce an invariant embedding of nodes and edges; (ii) a transformer model that produces an embedding of the edges of each tunnel in a manner invariant to the order of edges in the tunnel; and (iii) a recurrent unit to predict and iteratively improve the total traffic to be carried on each tunnel. The recurrent unit models the iterative approach used in optimization solvers starting with candidate solutions and making incremental adjustments to improve the objective function.

We evaluate HARP using data from (i) a private WAN (AnonNet) which includes a snapshot of the topology and traffic matrices over a multi-week period; (ii) data from networks for which publically available traffic matrices are available[1, 45, 46, 54]; and (iii) publically available larger scale topologies [33] using synthethic traffic data. Our key results are:

- Our analysis of AnonNet indicates significant topology variation (addition and removal of nodes and links, changes in edge nodes, variations in link capacity, recomputation of tunnels) over time owing to both natural topology evolution, as well as failures and planned maintenance.
- HARP is effective at handling topology variations or the AnonNet dataset. Even in the most challenging setting HARP achieves an MLU at most 11% higher than optimal for 98% of the snapshots, when trained on a small set of snapshots and tested on a wide range of snapshots that contain noticeably different topologies and sets of tunnels relative to the training data.
- The design elements HARP introduces are important to its performance. HARP outperforms TEAL and DOTE (state-of-the-art ML-based TE schemes) in environments where link capacities change, and when unseen perturbations to topology are encountered in the testing data (e.g., change of tunnels, unseen partial and full link failures). For instance, under unseen failure HARP achieves a 99.9^{th} percentile MLU of at most 1.09 of the optimal for GEANT. However, for DOTE and TEAL, only 63% and 50% of test cases respectively are within 1.10 of the optimal. Further, an ablation study confirms the importance of the recurrence unit in HARP which ensures alignment to the optimization algorithm.
- HARP outperforms Gurobi [20], a widely used optimization solver by (i) achieving lower MLU on the true traffic matrix when both methods are provided predicted traffic matrices – for three popular prediction methods, HARP achieves a median reduction in MLU of 5 to 10% and (ii) reducing computation time, achieving over an order of magnitude reduction for the KDL topology.

Overall, our results indicate the feasibility of designing ML-based TE schemes that can perform well despite topology variation, and indicate the importance of the mechanisms introduced by HARP.

2 BACKGROUND AND MOTIVATION

2.1 Background: ML-Driven TE

Given a network topology, a traffic matrix, and a set of tunnels (paths), traffic engineering schemes determine how traffic must be routed on each tunnel while optimizing a desired objective of interest. A common widely used metric is Maximum Link Utilization (MLU) [30], or the utilization of the most congested link. Traditionally, the allocation problem is solved using conventional optimization solvers [20].

Recent work [48, 61] explores the use of neural networks to solve optimization problems in the context of wide-area network traffic engineering as we discuss below:

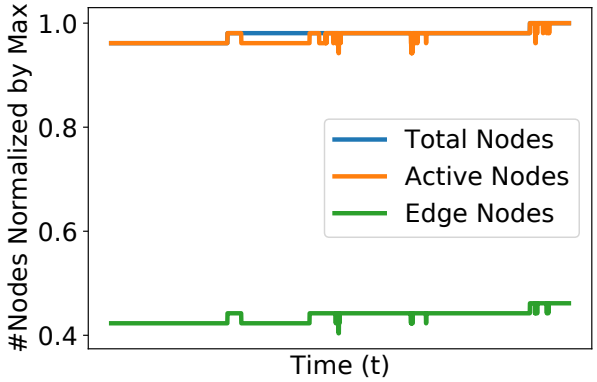
DOTE. Given a set of traffic matrices as input, DOTE determines how traffic must be routed on pre-determined tunnels for a future matrix. Thus, DOTE combines the task of predicting a future matrix, and computing optimal routes for the predicted matrix. DOTE achieves this goal using a simple feedforward deep neural network (DNN), also known in ML terminology as a Multi-Layer Perceptron (MLP). DOTE does not model nodes, edges, link capacities, tunnels and the relevant associations.

TEAL. Given a traffic matrix, TEAL [61] determines how to route traffic on tunnels for that matrix. TEAL represents topology using alternating layers of a Graph Neural Network (GNN) and a DNN. The GNN layer models the relations between edges and tunnels using a bipartite graph, while the DNN layer captures the interaction between tunnels associated with the same flow. Using the resulting embeddings associated with each tunnel of every flow, reinforcement learning is used to compute an allocation of how much traffic goes on each tunnel. For scalability, TEAL only takes the local information for each flow (i.e., the embeddings of the tunnels of that flow), and generates the intended split ratios. It relies on a final step which computes a global metric (e.g., link utilizations) while considering the allocations across all flows, which is the primary step where the interaction across flows is modeled.

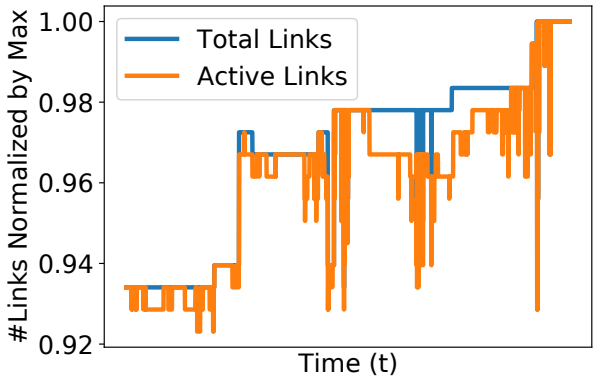
2.2 Challenges for ML-Driven TE

While these works open up an exciting line of inquiry, several major open questions need to be addressed for ML-driven TE to be a reality as we discuss below:

Handle changes to topology and tunnels is critical. Even in a single WAN context, any ML model must inherently be designed to capture the fact that the topology itself may vary over time. Figure 1 presents results showing the variation in the number of nodes and the number of links over a 4 week period in AnonNet. The figure shows a few key trends. First, the topology organically evolves with the total number of nodes and links (*Total Nodes* and *Total Links*) differing between the start and end of the time period. Such organic growth may occur due to the creation of new data centers, addition of new links and routers, or decommissioning links for costs and efficiency. Second, owing to planned maintenance events, or unplanned failures [17, 18, 40, 50, 53], the number of *Active Nodes* and *Active Links* is lower than the total and varies across time as well. Third, the set of *Edge Nodes* (where traffic ingresses/egresses the network) also varies across snapshots. Fourth, even when a link is active its capacity could vary with time owing to partial failures (e.g., a link may comprise multiple sub-links, only some of which may fail), and we discuss this further in §5. Finally, many of the above changes to topology necessitate the recomputation of tunnels prior to ML-based routing decisions.



(a) Variation in the number of nodes per snapshot (normalized to the maximum number of nodes across snapshots).



(b) Variation in number of links per snapshot over time (normalized to the maximum number of links across snapshots).

Figure 1: Changes in the network topology of a real WAN over a 4-week period.

Handle topologies and tunnel configurations outside the training set. An ML model may not have training data associated with changes such as addition of new routers and links (and changes to tunnels). Likewise dealing with failures poses challenges since it may not be possible to anticipate and train for all possible failure modes. Further, these changes may involve a recomputation of the set of tunnels. Yet, there is significant training data with slightly different topologies prior to the additions or failures that can be leveraged. Thus, ideally, ML-driven TE must be able to handle topologies (and the set of tunnels) different than those at training and leverage data available with slightly different settings.

Invariant to common input transformations. A starting point to ensure the learning algorithm can generalize across different topology and tunnel configurations, is to ensure the model by design can handle simple input transformations. For example, a model which encounters an identical network that it has been trained on, but with nodes, links and tunnels relabeled, should by design produce the same result. As another example, the objective value for optimal TE does not typically change when given the transpose of a traffic matrix. This is because network links often have symmetric capacity in both directions, and tunnels for (j, i) source-destination pair are the reverse of those for (i, j) pair. Thus, a model for ML-based TE must by design guarantee that it produces the same result for a matrix and its transpose.

2.3 Where existing solutions fall short?

Existing ML-based TE schemes fall short of the above desired requirements as we elaborate below:

Some schemes are designed for a fixed topology. As mentioned earlier, DOTE does not model nodes, edges, link capacities or associations between tunnels and edges. Its DNN (feedforward network) could be viewed as a function that maps a traffic matrix to how much traffic must be carried on each tunnel (i.e., split ratios). In general, allocation should depend on capacities, specific topology, choice of tunnels, and demands. Since DOTE's DNN only models demands, it is not clear how changes in other inputs can be accommodated.

Do not guarantee invariance to input transformations. DOTE's DNN is sensitive to the order in which traffic matrix entries are supplied. Even with the topology fixed, consider a scenario where it has been trained on traffic matrices in the training set, but provided the transpose of the matrices in the test phase. Such data cannot be handled as the DNN treats matrix entries in a positional manner. More generally, it is hard to predict the output under other graph transformations (e.g., relabeling nodes, edges and tunnels) making the scheme vulnerable to poor performance with unseen topologies (§5).

TEAL has alternating GNN and DNN (feedforward network) layers as discussed earlier. While GNNs do ensure invariance to simple topology transformations (e.g., relabeling of nodes), unfortunately, its DNNs are not invariant to changes in the ordering of tunnels. In particular, each DNN layer in TEAL concatenates embeddings of tunnels associated with a given flow as input and generates updated embeddings. Likewise, the RL module concatenates the embeddings of tunnels associated with a flow and generates split ratios. Doing so makes TEAL sensitive to the order in which the tunnels are supplied as inputs. If a tunnel recomputation shuffles the order in which this concatenation occurs (e.g., relabel tunnels between training and testing), this creates an embedding that the DNN layers as well as the RL module may not have seen in training.

Do not work well on topologies outside the training set. The issues above are in part what makes existing schemes vulnerable to poor performance when topologies outside the training set are encountered. In §5, we present extensive results demonstrating these issues.

3 HARP DESIGN

In this section, we present HARP, a novel algorithmic-aligned hypergraph neural network architecture for TE optimization. We present the principles motivating HARP, and elaborate on its design.

3.1 HARP design principles

HARP is explicitly designed to allow for changes to topology, and tunnels over time, rather than designed for a fixed topology. Further, in designing HARP, we seek to ensure key invariances to better allow the model to handle topologies outside the training set. To

Scheme	Models topology	Invariant to: Node relabeling	Invariant to: Tunnel reordering	Aligned Arch
DOTE	✗	✗	✗	✗
TEAL	✓	✓	✗	✗
HARP	✓	✓	✓	✓

Table 1: Comparing design elements of HARP with existing approaches.

this end, HARP is characterized by two key design principles: (i) ensure permutation invariance to topology and tunnel inputs; and (ii) alignment with TE optimization models.

Principle 1 (Ensuring permutation invariances). HARP targets two natural invariance properties: (a) permutation invariance to the order that the tunnels are given as input; (b) permutation invariance to joint permutations of node ids in the network topology and traffic demands; (c) permutation invariance over the order of the edges in the tunnel. The invariances ensure that HARP's outputs are robust to the reordering of nodes, the corresponding traffic demands, and tunnels, such that HARP's predictions are provably based on the structural and functional properties of the network, the traffic demands, and the tunnels rather than on specific node and tunnel identities. This makes the neural network output transferable across varying topologies, traffic demands, and tunnel reconfigurations.

Principle 2 (Aligning neural architecture to optimization model). The algorithmic alignment of HARP is rooted in neural algorithmic reasoning [57], a paradigm that seeks to endow neural networks with the ability to learn algorithmic patterns from examples. Our algorithmic alignment allows HARP to understand when its outputs must be adapted to conditions that could be evolving and *do not* mirror the conditions observed in the training data. In our training data, such adaptations happen to a lesser extent, which allows the algorithm to learn the rules needed to adapt to new scenarios.

HARP vs existing approaches. Table 1 presents results comparing HARP with DOTE and TEAL. DOTE's use of a simple DNN does not model many key aspects of network topology as discussed in §2. While TEAL does capture key features of topology, and its use of a GNN ensures invariance to relabeling of nodes, it does not ensure invariances to tunnel ordering, or align its policy for deciding split ratios with the optimization model. In contrast, HARP models topology, handles key invariances, and uses an aligned architecture.

3.2 HARP design overview

We next present a high level overview of HARP's design shown in Figure 2. We summarize the key components of HARP below, and elaborate in the following sections.

1. Network topology embedding for invariant embedding of edges (§3.3). HARP starts with a small GNN module (using a standard GNN [32, 42, 52, 59]) that takes the topology – nodes (routers), edges (links) and link capacities as inputs, and generates initial embeddings of all edges.

2. Invariant embedding of tunnels through a position-free transformer (§3.4). Next, HARP uses a *set transformer* module denoted SETTRANS, where the input is a tunnel represented by a set of edge embeddings derived from the GNN edge embeddings. The set transformer is crucial to modeling the invariant embeddings of tunnels as we elaborate later.

3. Tunnel initial split rate predictor. Next, a DNN module (MLP1) takes the output of the tunnel embedding and its associated demand and produces a guess of the initial split ratios. The same MLP1 is applied to each tunnel separately, complying with Principle 1(a).

4. Recurrent adjustment unit (RAU) to iteratively improve the optimization objective (§3.5). Finally, Figure 2 illustrates the RAU module of HARP that strives to meet Principle 2. Much like traditional optimization solvers iteratively improve a candidate solution, the *recurrent adjustment unit* (RAU) takes the embeddings of tunnels along with traffic matrices, and iteratively refines the split ratios (i.e., how much traffic is carried on each tunnel of every flow) while optimizing a desired objective. The key advantage over a standard optimization solver is the RAU's ability to make large adjustments in a single step.

We close with two comments. First, the architecture of our neural network, while it appears large, is efficiently structured around only four core modules that are repetitively applied over the data: 1×GNN, 1×SETTRANS, 1×MLP1, and 1×RAU. In fact, in our AnonNet experiments, the model selected in validation for HARP has 21K parameters, while DOTE's best model has 1M parameters. This is because the same four modules described above are the only modules used throughout HARP's computations.

Second, while HARP's overall architecture is general, the details of the RAU module must be tailored to the specific objective being optimized. In this paper, we focus on Maximum Link Utilization (MLU), a widely used TE metric, for concreteness. However, we believe the recurrent unit can be adapted in future studies to incorporate other metrics described in the literature [10, 16, 24, 26, 34]).

3.3 Invariant embedding of edges

In what follows, we detail how the Graph Neural Network (GNN) module in Figure 2 obtains *edge embeddings*. Let $\boldsymbol{A} \in \{0,1\}^{|V|\times|V|}$ be the adjacency matrix of the topology. The GNN module first creates node embeddings from $\boldsymbol{A}$ and the link capacities $\boldsymbol{C} \in \mathbb{R}^{|V|\times|V|}$, where $\boldsymbol{C}_{ij} \geq 0$ for all $i, j \in V$ s.t. $A_{ij} = 1$.

We create features for each node $i \in V$ which are the total capacity of edges connected to the node $\sum_{j:\boldsymbol{A}_{ij}>0} \boldsymbol{C}_{ij}$, and its degree. The embedding of an edge $\boldsymbol{A}_{ij} = 1$, denoted $h_{ij} \in \mathbb{R}^r$, where $r \geq 4$ is a hyperparameter, is the sum of the GNN node embeddings of i and j, concatenated with the capacity $\boldsymbol{C}_{ij}$. The node embedding sum ensures an extra invariance, such that the embeddings of $h_{ij} \in \mathbb{R}^r$ and $h_{ji} \in \mathbb{R}^r$ are different only if $\boldsymbol{C}_{ij} \neq \boldsymbol{C}_{ji}$, else they are the same.

The resulting GNN edge-embedding architecture is equivariant. Define $\text{GNN}(\boldsymbol{A}, \boldsymbol{C}) \in \mathbb{R}^{|V|\times|V|\times r}$ to be the **sparse** tensor output of our edge-embedding module. Equivariance here means that if $\boldsymbol{A}$ is the adjacency matrix of the topology and $\pi \circ \boldsymbol{A}$ is a matrix with jointly reordered rows and columns by an arbitrary permutation π of the nodes (i.e., the graphs represented by $\boldsymbol{A}$ and $\pi \circ \boldsymbol{A}$ are isomorphic), then $\text{GNN}(\pi \circ \boldsymbol{A}, \pi \circ \boldsymbol{C}) = \pi \circ \text{GNN}(\boldsymbol{A}, \boldsymbol{C})$, that is, the embedding of each edge is not affected by the node reordering, where $\pi \circ \text{GNN}(\boldsymbol{A}, \boldsymbol{C})$ is the equivalent reordering over the sparse edge-embedding tensor. *The GNN equivariance and our edge*

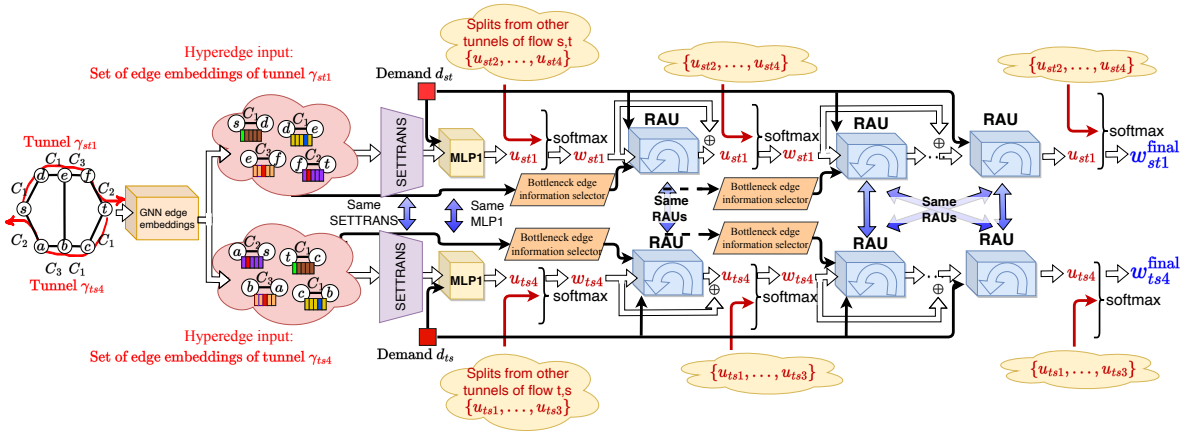


Figure 2: Tunnel's point of view of HARP's neural network archiecture. The figure assumes 4 tunnels per pair, and is illustrated with tunnels $st1$ (the first tunnel from s to t), and $ts4$ (fourth tunnel from t to s) as examples. All modules with the same labels (RAU, SETTRANS, MLP1) share the same parameters. Figure 13 shows a simplified version

embedding procedure guarantees compliance with the invariance in Principle 1(b).

We encode the topology into edge embeddings to take advantage of potential associations between TE and topology. For instance, edges with high capacity and high betweeness centrality—i.e., a good fraction of all shortest paths pass through that edge—are likely to share their capacity among many routes in the final TE solution and, hence, can aid HARP obtain good initial tunnel split ratios.

3.4 Invariant embedding of tunnels and tunnel-edge tuples

(a) Modeling the invariances in tunnels. In the TE context, the k-th tunnel for the flow $s \rightarrow t$ is a type of *hyperedge* in the network topology, defined through a *set of directed edges*, defined as $\gamma_{stk} \subseteq E$. Hyperedges extend traditional definition edges to encompass multiple nodes and edges. The network topology at the left of Figure 2 shows two tunnels overlayed over the network topology and their corresponding hyperedges as two sets of edge embeddings, respectively. More precisely, the sets are actually *multisets*, since edges are allowed to have the same embeddings and we want to keep the repetitions. We will refer to the multisets as *sets* to use a more familiar notation.

Hyperedge embeddings. The hyperedge γ_{stk} representing the k-th tunnel is embedded through a set transformer [36] (SETTRANS) over the embeddings of the edges in γ_{stk} given by the GNN module in Figure 2. The set transformer can also be replaced by a standard transformer model [56] without positional encodings. The same SETTRANS module in Figure 2 is applied to all tunnels.

A *set transformer* [36] is a neural network that takes a set as input and outputs an embedding for each element of the set. By describing tunnel γ_{stk} as a set of (directed) edge embeddings, as shown in Figure 2 (*Hyperedge input*), the set transformer is equivariant to permutations of the edges in the tunnel. That is, if the tunnel γ_{st1} in Figure 2 is described as a matrix $\boldsymbol{H}_{st1} = [[\text{CLS}], h_{sd}, h_{de}, h_{ef}, h_{ft}] \in \mathbb{R}^{5\times r}$, where $h_{**}, [\text{CLS}] \in \mathbb{R}^r$ is the edge embedding of each of the four edges, and CLS is a special vector, then $\text{SETTRANS}(\pi' \circ \boldsymbol{H}_{st1}) = \pi' \circ \text{SETTRANS}(\boldsymbol{H}_{st1})$ for any permutation π' acting on the rows of $\boldsymbol{H}_{st1}$ (i.e., a reordering of the edges in the tunnel). This equivariance complies with Principle 1(c) as SETTRANS generates equivariant embeddings for each edge conditioned on tunnel γ_{st1} (denoted *edge-tunnel* embeddings henceforth) and the corresponding BOS[1] (CLS token) embedding is the embedding of tunnel γ_{st1} itself.

As Figure 2 shows, the same SETTRANS is applied to all tunnels, complying with Principle 1(a). For instance, as tunnels γ_{st1} and γ_{ts3} in Figure 2 have the same set of edge embeddings that will be the input to SETTRANS (same matrices $\boldsymbol{H}_{st1}$ and $\boldsymbol{H}_{ts4}$ up to a permutation of their columns), then these tunnels get the same corresponding edge-tunnel and tunnel embeddings as outputs of SETTRANS.

Finally, the demand matrix is a separate weight matrix $\{d_{st}\}_{s,t\in V}$. Here, the neural module in Figure 2, a DNN denoted MLP1, takes the embedding of each tunnel γ_{stk} and the demand of the relevant flow d_{st} and outputs a proposed (unscaled) split ratio (u_{stk}) for that tunnel. Each tunnel is individually considered at this point, and we consider multiple tunnels of the same flow together later.

3.5 Recurrent Adjustment Units: aligning to the optimization algorithm

TE optimization algorithms typically involve an iterative refinement of TE solutions. To ensure better alignment, HARP uses a neural network architecture with recurrent processes that mimic this iterative procedure.

The *Recurrent Adjustment Units* (RAU) in Figure 2 illustrates HARP's architecture alignment. The RAU module is employed recursively and can be thought as a specialized Recurrent Neural Network (RNN) for TE optimization, where the number of recursions is a hyperparameter of HARP (our experiments use between 3 and 14 RAU recursions). The recursion ensures that RAU learns

[1] beginning of set/sequence in ML-terminology

how to iteratively improve a proposed solution. This aligns with how TE optimization algorithms interate over solutions, with the main difference being that RAU requires very few iterations. As we show in our ablation study (Figure 6), the RAU algorithmic-aligned recursion is key to HARP's ability to transfer what it has learned in training scenarios to significantly different test scenarios. RAU starts with the u variables output by MLP1 (the initial unnormalized split ratios) described earlier. A subsequent softmax layer takes the relevant u variables for all the tunnels from s to t and normalizes them to produce w variables. This ensures the sum of the appropriate w variables for each pair of endpoints s and t adds to 1 The w variables determine how much traffic go on each tunnel, but the cumulative traffic across all tunnels on any given edge may exceed the capacity of an edge. To address this, a sequence of RAUs penalizes capacity overruns and promotes better link utilization for underutilized links (given our focus on the MLU metric).

After the softmax is executed for all s-t pairs, the system computes (i) the Maximum Link Utilization (MLU) of the entire network; and (ii) for every tunnel γ_{stk}, the bottleneck link of that tunnel (l_{stk}), and its associated utilization $U(l_{stk})$. The bottleneck link of a tunnel is simply the link in the tunnel with the highest utilization. For every tunnel γ_{stk}, we feed (i) the network-wide MLU; (ii) the tunnel-edge embedding of the bottleneck link l_{stk}; (iii) $U(l_{stk})$, and (iv) the demand d_{st} into another DNN (RAU) individually. This step is pivotal: feeding bottleneck link embeddings allows RAU to discern which edge in the tunnel is most utilized and understand its capacity limits. Furthermore, RAU compares the network-wide MLU with $U(l_{stk})$. If these values are equal, the split ratio of this tunnel should be reduced as it impacts the network-wide MLU. Conversely, if the network-wide MLU exceeds $U(l_{stk})$, then the tunnel does not currently impact the network-wide MLU and its traffic allocation can be increased.

Subsequently, RAU proposes split ratio adjustments that supplement the existing ratios, resulting in updated split ratios at the end of each RAU. The architecture leverages a common RAU function across all tunnels, fostering a policy invariant to tunnel IDs and adaptable to demand changes. The adjustment of a previous RAU is added to the current split ratios and then fed into the next RAU or, if at the last RAU, to the final softmax, giving the final split ratios of HARP.

4 EVALUATION METHODOLOGY

Our evaluation is driven by the following **goals**:

• What kind of topology changes occur in real-world environments that are important for TE schemes to consider?

• Is it feasible for an ML-driven TE scheme to perform well in a dynamically evolving topology environment (§2), where the set of nodes, links and tunnels change with time, and the capacities of links also change?

• How important are the new design mechanisms introduced by HARP in ensuring ML-based TE schemes can work well in real-world environments where topologies vary over time?

• What advantages does HARP provide relative to traditional optimization solvers from the perspective of (i) computation costs; and (ii) performance on a future traffic matrix when only traffic matrices of the recent past are available?

We explore these questions by evaluating HARP using real-world datasets, and through comparisons with state-of-the-art ML-based TE schemes.

Datasets. Our experiments are conducted using a dataset from AnonNet, a large private WAN (§2) with several tens of nodes and several hundred edges. The data includes snapshots at 1 second intervals over a multi-week period of the topology and the traffic matrices for each snapshot. As shown in §2, the topology continually evolves during the dataset (owing to natural growth and failures) with noticeable variation in the total and active nodes and links over time. We complement data from AnonNet with publically available real-world datasets for GEANT and Abilene [1, 45, 46, 54] which include traffic matrices at different snapshots (the topology is unchanged throughout the entire period). To evaluate HARP's ability to scale to larger topologies, we conduct experiments on the KDL topology [33] and synthethic traffic data. To evaluate HARP's ability to handle topology variations in these datasets, we perturb the topology in multiple ways – e.g., reordering tunnels, and through different failures not seen in training (§5.5).

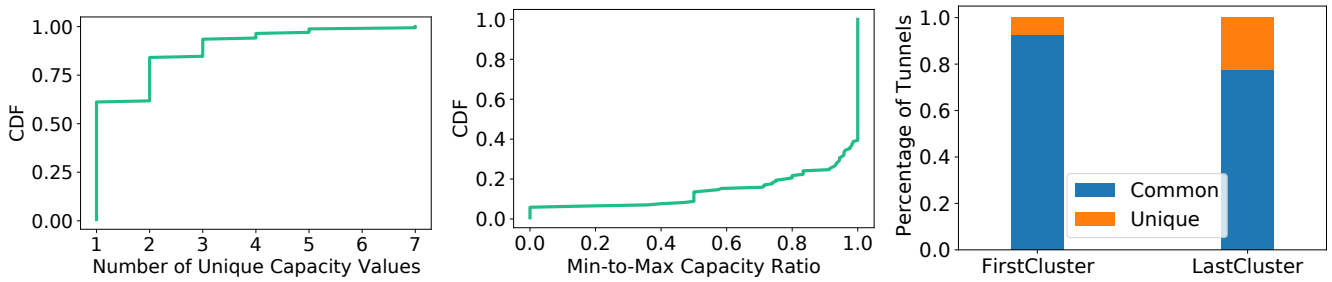
Metrics. We focus on MLU, a widely studied metric for TE. Since our focus is on evaluating how close to optimal different ML-driven TE schemes perform, we present Normalized MLU in all our plots (henceforth abbreviated as **NormMLU**) which is the ratio of the MLU with a ML-driven TE scheme to the MLU of optimal (obtained using the Gurobi optimization solver).

HARP. We used PyTorch[47] and PyTorch Geometric[13] to build HARP following Figure 2 and Section 3, and present details in the Appendix (Appendix A.1). We used a standard transformer [56] without positional encoding to implement SETTRANS. We also employed the Adam optimizer [31] for training. HARP recomputes split ratios for each snapshot. Our experiments indicate that in the case of link failure, HARP automatically ensures no traffic is carried on unavailable tunnels owing to the recurrent unit (§5.3) which iteratively adjusts split ratios. Hence, we did not employ rescaling in our experiments with HARP.

Schemes compared: To highlight the importance of HARP's design mechanisms to handle topology variations, we compare HARP with DOTE and TEAL, which are representative of state-of-the-art in ML-based TE using the code provided by the authors for both systems.

DOTE. DOTE is originally designed to take a series of TMs at time steps $t_0, t_0+1, \ldots t_0+h-1$, and predict the routing for a TM at time step t_0+h. We instead modify it to take a single TM and predict the routing for that TM given our focus is on how effectively ML approaches can handle variations in topologies. DOTE is designed for a fixed topology, and does not explicitly recompute split ratios when a link fails. Consequently, the split ratio predictions from DOTE may result in traffic traversing the tunnels that traverse the failed link. To remedy this, all our results with DOTE are presented assuming a local rescaling approach when dealing with a complete link failure following the author recommendations [48]. That is, for any flow, traffic on unavailable tunnels is rerouted on the surviving tunnels in a manner proportional to traffic distribution in the absence of link failure.

TEAL. Although TEAL [61] directly recomputes routes instead of rescaling, our experiments with the author-provided TEAL code indicated that it was not always able to move traffic away from



(a) Number of unique capacity values per link across snapshots of the cluster. **(b) Ratio of min to max capacity of link across snapshots of the cluster.** **(c) Comparing the set of tunnels in the first and last cluster.**

Figure 3: Variation of link capacities for one of the larger clusters of the AnonNet dataset.

a failed link resulting in an MLU of ∞ under link failure in many cases.[2] Consequently, we applied a similar rescaling adjustment on complete link failure - for all flows, we ensured all traffic was carried on surviving tunnels with the proportion of traffic on tunnels being maintained the same.

Hyperparameter search. On every dataset and for each system, we conducted an extensive grid search to tune hyperparameters. For HARP we searched for the number of GNN layers, the number of SETTRANS layers, the number of RAU iterations, the batch size, and the learning rate. For TEAL, we tuned the batch size, learning rate, the number of samples to estimate the reward, the pseudorandom number generator seed (important for DeepRL [9, 21]), and the number of FlowGNN (GNN-DNN) layers. For DOTE, we tuned the learning rate and the batch size. For each hyperparameter setting, (i) we trained for sufficient epochs until the systems converged or stopped; and (ii) saved the model after every epoch. Given N combinations of hyperparameters and M epochs, this resulted in upto NM models totally of which we picked the best on the validation set. For example, for the GEANT dataset, we explored 72 different sets of hyperparameters for TEAL, 24 for HARP, and 6 for DOTE due to its simplicity and the little effect they had in DOTE's results. For each system, and each dataset, we did a grid search for every set of hyperparameters. Appendix A.2 shows a full list of hyperparameters we searched.

5 RESULTS

We begin by characterizing the topology dynamics of the AnonNet dataset (§5.1). We next evaluate HARP using the AnonNet dataset (§5.2) to explore the viability of ML-based TE in such settings. We evaluate the importance of the new mechanisms introduced by HARP to handle topology variations through comparisons with existing ML-based TE schemes (§5.2, §5.4, and §5.5), and an ablation study(§5.3). We evaluate HARP's ability to handle perturbations in topology outside training data for a large scale topology (§5.4) and using datasets with real traffic matrices (§5.5), and report on computation times(§5.6). Finally, we evaluate HARP when provided predictions of future traffic matrices (§5.7).

[2]The TEAL [61] implementation is extensively optimized for the Maximum Flow metric for which it includes adjustments to ensure link capacity constraints are not violated. These optimizations were not implemented for the MLU metric, which is our focus.

Tunneling scheme. Our experiments with AnonNet used 15 shortest paths to ensure i) the graph remained connected when considering complete link failures and ii) the optimal MLU is not much higher than 1. We used 4 shortest paths for experiments on KDL following prior work [61], and used 8 shortest paths by default for all other topologies, in line with prior work [48].

5.1 Nature of real-world topology changes

We start by presenting more analysis of the AnonNet dataset to discuss the types of topology changes that ML-based TE must handle in practice. Recall that each topology snapshot includes (i) the set of total nodes and links; (ii) the set of active nodes and links at that snapshot; and (iii) capacities of the links. We group contiguous snapshots into clusters based on the nature of topology variation across the snapshots as we discuss below.

Topology variations across clusters. A snapshot is assigned to a new cluster when there is a change in the set of Active Nodes, when a new link is added, or when there is a change in the set of Edge Nodes (which correspond to nodes where data ingresses or egresses the network). A snapshot is otherwise assigned to the same cluster. While all snapshots in the same cluster have the same set of tunnels, it is natural to recompute tunnels across clusters given the addition of nodes and links, and changes in the set of edge nodes. Overall, we grouped all snapshots in the dataset into 78 clusters. Figure 3c compares the set of tunnels of the first cluster (FirstCluster) and last cluster (LastCluster). The results show that 20% of tunnels in LastCluster were added and not present in FirstCluster, and 8% of tunnels in FirstCluster were no longer present in LastCluster.

Topology variation within a cluster. We note that the capacities of links may still show significant variation within a cluster. Figure 3 illustrates this variation in capacity within one of the largest clusters. For each link, we compute (i) the number of different capacity values observed over the cluster; and (ii) the capacity variation ratio, i.e., the ratio of the minimum to maximum capacity for that link across snapshots in that cluster. Note that we do not create a new cluster for a snapshot with a complete link failure.

Figure 3(a) shows that 40% of the links in the cluster exhibit multiple capacity values, with some links showing as many as 7 different capacity values in the same cluster. Variation in link capacity may occur because a link between nodes i and j may

comprise multiple sub-links, a subset of which may be temporarily inactive (planned maintenance or failures). Further, each sub-link in turn is an aggregation of multiple physical circuits, and the capacity could reduce if some of the physical circuits fail. Figure 3(b) shows that 20% of the links have a capacity variation ratio of 0.8 or less, with about 5% of the links having snapshots with zero capacity (the link is fully inactive owing to failure or maintenance). Note that for zero capacity links we we assign a small value of 1e-4 (significantly smaller than the capacity of other links) to allow flow of gradients when training ML-based TE schemes. The cluster had over 250 capacity configurations, where two configurations differ in the capacity of at least one link.

In the Appendix, we present the variation in link capacity over the complete dataset (Figure 15). 80% of links witness more than one capacity value with some links having 33 unique values. Further, 20% of links have at least one snapshot where they are completely unavailable, and 60% of links have a min-to-max ratio of 0.8 or less.

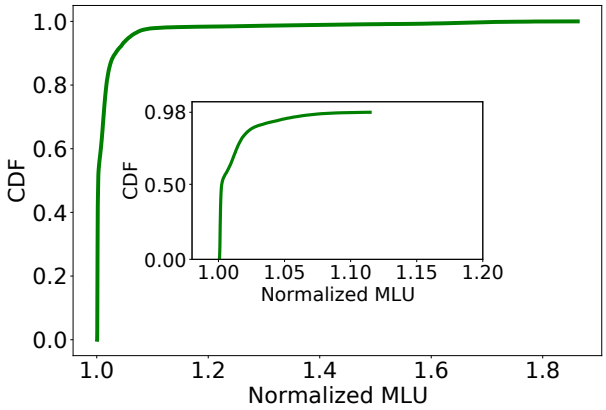


Figure 4: HARP performance in the most challenging setting where it is trained on three clusters and tested on 72 other clusters spanning the entire AnonNet dataset. The inner figure is a zoomed version truncated at the 98th percentile.

5.2 Can HARP handle topology changes?

We next evaluate whether it is feasible for ML-based TE to perform well in real-world environments where topologies change over time using the AnonNet dataset. To this end, we train HARP on samples from the first three clusters, validate it on the next three clusters, and test it on over 20K snapshots belonging to all remaining 72 clusters. Figure 4 presents a CDF that shows the NormMLU achieved by HARP across snapshots. HARP performs well – 98% of the snapshots achieve a NormMLU of under 1.11, while even the most challenging snapshots out of 20K test cases have a NormMLU of 1.86 or less.

We have further stressed HARP through experiments where HARP is trained (and validated) on each of the first three clusters separately and tested on the same 20k snapshot test set. HARP continues to perform well with only a slight degradation. In all cases, when training with a single cluster and testing on the rest, 95% of the snapshots see an MLU of under 1.12 (compared to more than 98% when training with three clusters together) although the maximum degrades more (see Appendix for details).

Overall, the results highlight HARP's **transferability**, and its ability to work well on topology variants that it has not seen in training data. That is, HARP can generalize to other clusters even when trained with data from a very small number of clusters, and its performance gracefully improves as training data of more clusters is provided.

Benefits of explicitly modeling topology. We next explore the importance of the new design elements introduced by HARP in achieving the above performance by comparing it with DOTE. We also tried evaluating TEAL on the AnonNet dataset but despite an extensive hyper-parameter search, we were unable to get the training process to converge (we elaborate further in the Appendix).

While HARP explicitly models the network topology and link capacities, DOTE is designed for a fixed topology, a fixed-sized traffic matrix, and a fixed set of tunnels. DOTE does not model link capacities, and produces split ratios that are only a function of traffic demands. To illustrate the benefits of explicitly modeling topology, we compare HARP and DOTE in a more constrained setting where the systems are trained and tested on snapshots within the same cluster. We focus on single cluster settings since it is unclear how to easily adapt DOTE to settings involving multiple clusters (since the set of nodes, links, and tunnels, and the size of the traffic matrix vary across clusters).

Figure 5 compares HARP with DOTE for three of the largest clusters in the AnonNet dataset. In every experiment, 75% of the snapshots were assigned for training, 12.5% for validation, and 12.5% for testing. Each figure corresponds to a cluster and shows a CDF of the NormMLU obtained with each of the schemes. The figure shows that HARP performs very well with a maximum NormMLU of only 1.13, 1.02, and 1.07 for the three clusters respectively. In contrast, DOTE does not generalize as well. With DOTE, even the median NormMLU across snapshots is 1.12, 2.12 and 2.79 for the three clusters, with the maximum values being 2.03, 4.02 and 3.35.

The results stem from the fact that although link capacities vary across snapshots, the DOTE model is constrained to learning a single set of split ratios that work reasonably well for multiple capacity configurations for a fixed traffic demand. In contrast, HARP can adapt split ratios based on the capacity configuration. Second, DOTE is not designed to handle capacity configurations outside the training set unlike HARP which seeks to do so by meeting key invariances (Sections 3.3 and 3.4), and using an aligned architecture (Section 3.5). Finally, the NormMLU with DOTE is higher for the two clusters on the right – we hypothesize this is because these clusters had relatively few snapshots with complete failure in the training set, while the training data for the left most cluster had more of a balance of snapshots with and without complete link failure scenarios.

5.3 Benefits of HARP's aligned architecture

The marked effectiveness of HARP can be, in part, attributed to its architecture's alignment with TE optimization algorithm (Section 3.5). This alignment is evident in HARP's RAU (recurrent adjustment unit), which parallels the iterative refinement process

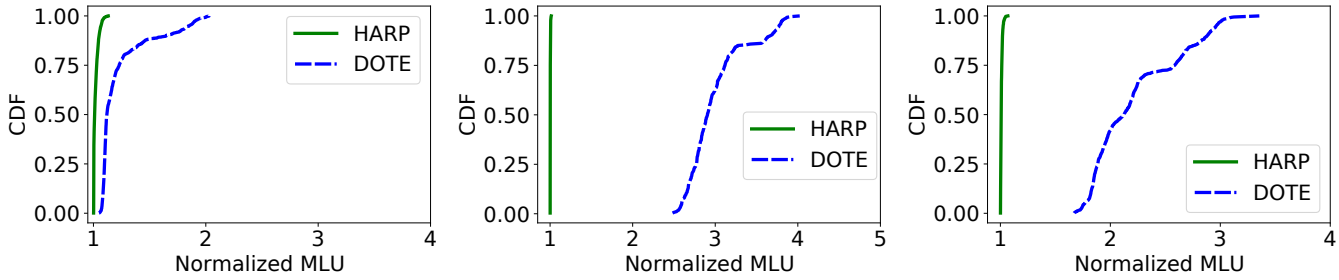


Figure 5: Comparing HARP and DOTE in settings where link capacities vary across snapshots but topologies are otherwise the same. Results are presented when training and testing on the same cluster for three different AnonNet clusters.

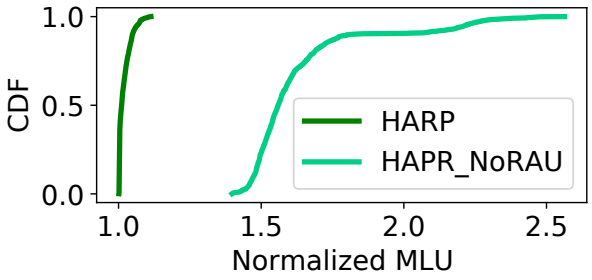


Figure 6: Ablation comparing HARP and HARP-NoRAU.

of TE optimization solvers, where an initial solution is progressively enhanced through a series of iterations that systematically converges towards an optimal or near-optimal solution by improving upon the initial solution. To evaluate the importance of RAU, we conduct an ablation study that compares HARP with a variant (HARP-NoRAU) that does not involve a recurrence unit.

Recall from §4 that HARP does not employ a local rescaling mechanism. However, we found that a noticeable tail with high MLU in our experiments with HARP-NoRAU in the absence of rescaling. To handle this, we report results with HARP-NoRAU with a rescaling policy (similar to DOTE and TEAL). Figure 6 compares HARP and HARP-NoRAU when training and testing with one of the largest clusters in the AnonNet dataset. HARP improves the median NormMLU from 1.56 to 1.01. This significant improvement indicates the importance of the recurrence unit in improving the optimization objective. The fact that HARP does not require rescaling also results from the effectiveness of its RAU unit in moving traffic away from a failed link. Overall, the results highlight the benefits of HARP's aligned architecture.

5.4 Handling perturbations on a large topology

While our results so far have focused on AnonNet given access to real topology evolution and traffic matrix data, we next present additional results on KDL [33], a large scale topology. We evaluate the ability of HARP, DOTE and TEAL to generalize from small topology changes by training on the original topology but perturbing the topology in the testing phase in two ways: (i) using a different set of tunnels than used in training; and (ii) creating partial failure scenarios where the capacity of some links is reduced relative to training. We generated 278 synthetic traffic matrices using the code provided by [48], of which 170 were used for training, 30 for validation, and the rest for testing.

Invariance to tunnel ordering. HARP has been explicitly designed to be invariant to the order in which tunnels are provided by input, a necessary condition to ensure the model can transfer to testing data that involves a different set of tunnels that seen in training. To evaluate this, we conduct an experiment where in the testing phase, we shuffle the order of the tunnels for each source destination pair using a different order from the one used during training. Figure 7 (right group of bars) show the performance of different schemes when tunnels are shuffled in such a manner. For comparison, the left group of bars presents the performance of schemes when the tunnels are provided in the same order in both training and testing for all schemes.

We make two observations from Figure 7. First, when the ordering of tunnels is preserved (left), all schemes perform well achieving almost ideal MLU across all matrices in the testing set. However, when tunnel order is shuffled (right), HARP retains its performance while both TEAL and DOTE degrade. This result stems from the fact that HARP is intrinsically designed to ensure invariance to tunnel ordering.

Handling partial failures Next, we train each ML-based TE scheme on the original topology, but test them on topologies with partial failure scenarios. Each partial failure scenario involves selecting a single link at random, and reducing its capacity by a value selected randomly between 50% and 90%. We generate 40 such partial failure scenarios, and test across all combinations of traffic matrices in the testing set and the 40 scenarios. Figure 8 shows a CDF of the NormMLU across the different combinations. Across all cases, HARP achieves NormMLU of less than 1.09, while the 75th percentile of DOTE and TEAL are 1.46 and 1.48 respectively. A key reason for the out-performance is that HARP can move traffic away from tunnels that traverse links with lower capacity owing to its RAU unit. In contrast, DOTE and TEAL are unable to do so. The performance degradation is especially severe when a link with higher utilization experiences partial failure and as the degree of failure increases.

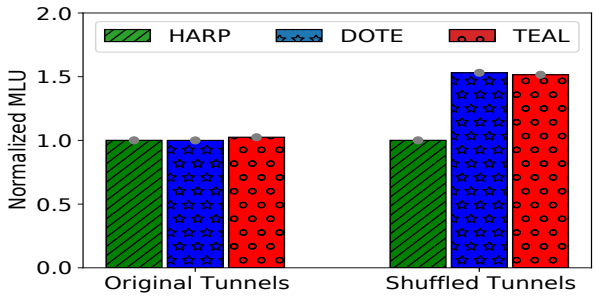


Figure 7: Comparing ML-based TE schemes for KDL topology when (a) tunnels in training and testing are in the same order (left group); and (b) when the order of tunnels is shuffled in testing relative to the order used in training (right group). Each bar shows the average NormMLU across the testing set, and errors bars show the standard deviation. Note the variance is low for all schemes

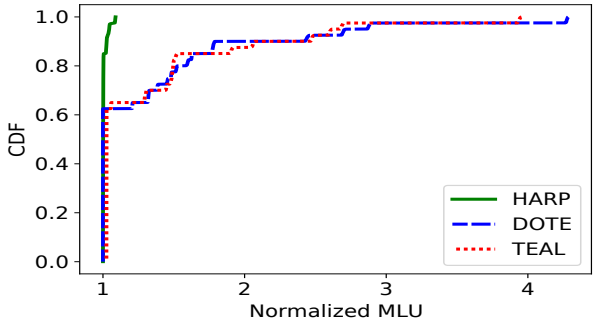


Figure 8: Comparing ML-based TE schemes trained on the original KDL topology and tested under partial failures

5.5 Performance on other datasets

We next present results on the GEANT and Abilene, for which traffic matrix data is publicly available. Here we evaluate the ability of HARP, DOTE and TEAL to generalize from small topology changes by training on the original topology (without failure) but testing the topology under different single link failure scenarios.

We train all systems on the original topology (without failures) using 75% of the traffic matrices over a two week period for GEANT and an eight week period for Abilene. The remaining 25% was split evenly between validation and testing sets. For each matrix in the testing set, we evaluated each system on every possible scenario involving the complete failure of a single link.

The top figure shows the performance of HARP. Each point on the x-axis corresponds to a particular link which failed, and the corresponding boxplot shows the NormMLU obtained with HARP for that failure scenario. Each boxplot shows the distribution of NormMLU across traffic matrices. Across all failure scenarios, HARP's median NormMLU ranges from 1.0 to 1.02, while its maximum NormMLU ranges from 1.0 to 1.17.

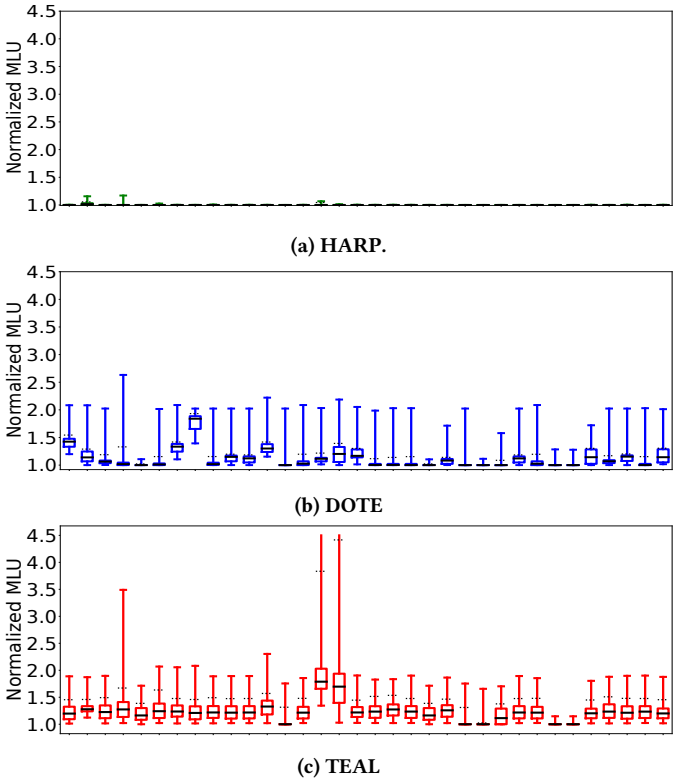


Figure 9: NormMLU of HARP, DOTE and TEAL under different link failure scenarios for GEANT where each boxplot represents distinct single-link failure. The top whisker goes up to the maximum, while the dashed lines show the 90th percentile.

Figure 9(b) (the middle figure) shows the results with DOTE. The performance is clearly not as effective under link failures because rescaling is not as effective as recomputing routes. DOTE's median NormMLU ranges from 1.0 to 1.48 across failure scenarios, while its worst case NormMLU ranges from 1.0 to 2.126 across failure scenarios. Figure 9(c) (lowest figure) presents results with TEAL. We use rescaling with TEAL as well as we found it resulted in an MLU of ∞ without rescaling for reasons outlined in §4. The results show TEAL's MLU computations do not perform as well under failures (as discussed in §4[2]).

Figure 10 presents a similar comparison of the three systems for Abilene. Again, we trained on the original topology without failures and considered all combinations of traffic matrices and single link failure scenarios in the testing phase. The graph shows that HARP significantly outperforms DOTE and TEAL, achieving a median and worst-case NormMLU of 1.0 and 1.33 respectively. Overall these results confirm HARP's ability to effectively handle topology perturbations not observed in the training data.

5.6 Computation Time

Figure 11 compares the computation time for HARP with other ML-based systems and Gurobi [20]. The time for all ML-based systems

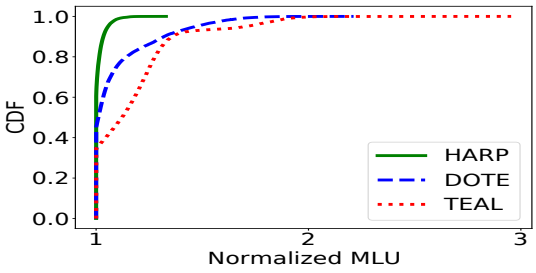


Figure 10: Comparing HARP with DOTE and TEAL on Abilene when training without failures, and testing on failure scenarios. Each CDF shows performance over all combinations of matrices in the testing set and single link failure scenarios. See Appendix (Figure 17) for boxplots showing breakdown per failure scenario.

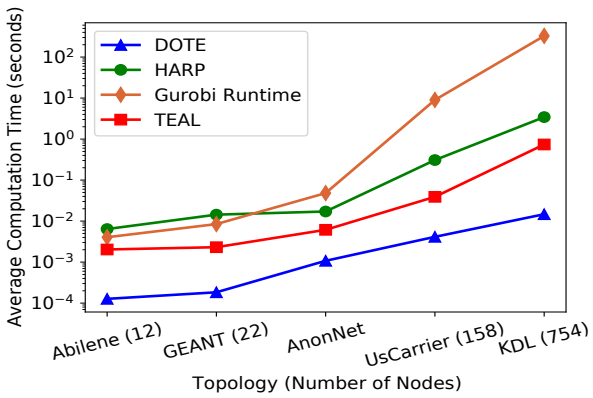


Figure 11: Comparison of computation times

was obtained over an NVIDIA A100 80GB PCIe GPU. For Gurobi, the runtime was obtained with an AMD EPYC 7763 64-Core processor with 128GB RAM, a more powerful configuration than previously reported in the literature [2, 61]. Besides the earlier datasets, we test HARP's ability to scale using additional large topologies from the Internet Topology Zoo [33] with synthetic traffic data. Like earlier, our timing experiments assumed 8 tunnels per flow for all topologies, except AnonNet which used 15, and KDL which used 4.

The results show HARP achieves significant improvements over Gurobi. For the largest KDL topology, Gurobi sees a reduction of runtime that is more than an order of magnitude. While TEAL and DOTE see even lower computation times, we believe the recomputation times of HARP are acceptable in practice. Further, the much better MLU achieved by HARP, and its more robust performance to topology variations, make HARP a good design point when balancing accuracy and computation times. Finally, we note that the HARP code is not optimized for computation times, and a variety of optimizations can be implemented in the future such as using a set transformer[36] instead of a regular transformer[56], replacing the standard transformer attention mechanism with the accelerated flash attention mechanism[19], and 8-bit quantization, pruning, and distillation of the neural networks [11, 23, 41].

5.7 HARP with predicted traffic matrices

Our results so far have evaluated schemes with exact traffic matrices. However, in practice, both optimization solvers and ML based techniques work on predicted traffic matrices (TMs), and their effectiveness is impacted by the accuracy of the predictions.

We next evaluate HARP when given predicted matrices using the AnonNet dataset. We use the HARP architecture as such except that it is fed a predicted matrix. In the training phase, the split ratios are generated using the predicted matrix but the loss is computed using the true matrix. Note that in the inference phase, only the predicted matrix is available to HARP. We emphasize that this is a quick adaptation (which we refer to as **HARP-Pred**) to show the potential of HARP with predicted matrices - more general adaptations are possible in the future.

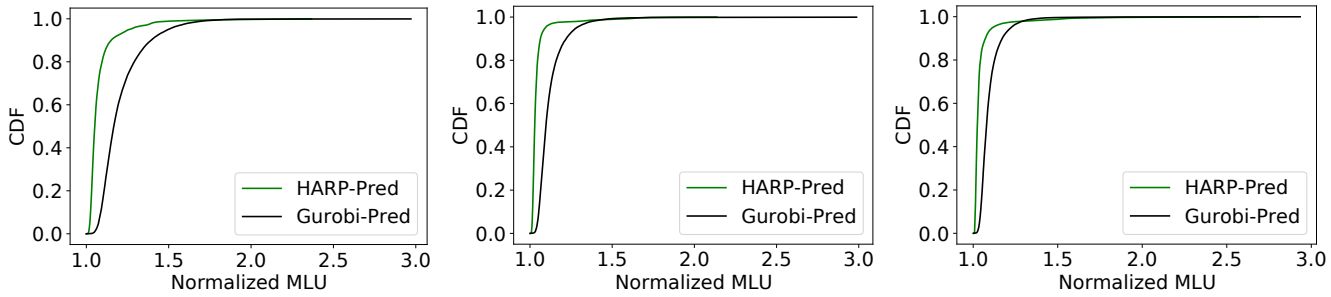
We evaluate HARP-Pred with multiple commonly used methods for predicting TMs. Specifically, we consider (i)*MovAvg*: for each cell, the predicted value is the average of the corresponding cells in the last 12 TMs; (ii) *ExpSmooth*: for each cell, the predicted value is obtained using an exponential smoothing scheme with a smoothing factor of 0.5; and (iii) *LinReg*: for each cell, the predicted value is based on a linear regression of the cell values in the last 12 TMs.

The *LinReg* parameters for each cluster were learnt using data from all previous clusters. We did not use the first cluster when training HARP-Pred since that was used for training *LinReg* parameters. We trained and validated HARP-Pred on the next few clusters, and tested on the rest.

We refer to **Gurobi-Pred** as a scheme that uses Gurobi on a predicted matrix. Note that Gurobi-Pred is optimal for the predicted matrix, but not for the true matrix. Figure 12 compares HARP-Pred and Gurobi-Pred for the three different predictors over the different snapshots. For both schemes, NormMLU represents the MLU normalized with respect to the MLU obtained on the true matrix for each snapshot.

The figure shows that HARP-Pred outperforms Gurobi-Pred for all three predictors. For instance, for *LinReg* (the best predictor), the median and 90%ile NormMLU achieved by Gurobi-Pred is 1.08 and 1.17 respectively. In contrast, the corresponding values with HARP-Pred are 1.02 and 1.07. The benefits are even more significant with other predictors. For instance, for *MovAvg*, HARP-Pred achieves a median NormMLU of 1.05 while the corresponding performance for Gurobi-Pred is 1.16.

An interesting open question is why an ML-based TE scheme such as HARP outperforms Gurobi when given predicted matrices. Our hypothesis and initial investigations suggest that the benefits accrue since HARP-Pred learns to be robust against prediction errors, although we defer a more systematic investigation to the future. Another open question is how the quality of predictor impacts the relative performance of Gurobi-Pred and HARP-Pred. Our preliminary experiments suggest that as the predictor gets weaker, while the performance of both degrade, HARP-Pred still outperforms. At the extreme, with an extremely weak predictor (e.g., one that predicts noise), HARP-Pred learns to ignore the predictor and develop a routing that performs not too poorly for any matrix in



(a) Moving Average Predictor (b) Exponential Smoothing Predictor. (c) Linear Regression Predictor.

Figure 12: Comparing HARP and Gurobi when given predicted traffic matrices on AnonNet for different predictors. The MLU is normalized with the optimal MLU on the true matrix.

the training set. In contrast, the routing that Gurobi-Pred produces with such a predictor effectively has no relation to the true matrix. At the other extreme, Gurobi does outperform for an ideal predictor. Here though, HARP performs close to optimal and can provide computation time benefits as our experiments with KDL show. That said, an interesting question is what the "crossover" point is – that is, at what levels of prediction accuracy does an ML-based TE scheme start providing benefits over classic optimization.

6 RELATED WORK

There is a rich history of work related to wide-area TE in general [3, 14, 49], and in the context of SDNs in particular [24–26] which have relied on classical optimization techniques. There has been much recent interest in using ML techniques for TE [38, 48, 55, 61] including works which use reinforcement learning for TE [15, 62, 63], and network planning [64]. Our focus is on neural approaches which can handle topology variants outside the training data through architectures that are invariant to common input transformations, and aligned with the optimization model.

Recent works estimate traffic matrices from link loads using deep learning techniques [29, 44, 60] including GNNs [39]. Others have used GNNs [51] to predict delay and jitter given topology, routing and traffic matrix. In contrast, we use GNNs to embed edges and ultimately tunnels in a network invariant fashion when solving optimization problems for TE.

There are many ongoing efforts to accelerate the solving of optimization problems. Researchers have attempted to scale optimization techniques [2, 16, 27] through decomposition techniques in the context of TE. However, the task remains challenging, and ML inference offers an alternative. First order methods [4] scale better but do not yield as accurate solutions and are still an active area of research [12]. Recent work [43] reduces the number of LPs that need to be solved to obtain fair solutions. However, it still uses Gurobi as the LP solver.

Many TE schemes handle transient failures by local rescaling, and proactively guarantee the network remains congestion-free over all failure scenarios in a specified set [28, 37, 58], or over sufficient scenarios to meet a desired percentile target [7, 8] by conservatively allocating bandwidth. Others [2, 27, 35] have argued for explicit recomputation. While local rescaling helps in the transient phase, it is not as performant as recomputing routes. More generally, recomputation may be required when the set of tunnels change, or when link capacities change.

7 CONCLUSION

In this paper, we have made two contributions. First, we have shown that it is feasible to design an ML-based TE scheme that can handle variations in topology that may be encountered in the real-world. Second, we have presented HARP [3], a neural model that achieves this goal. HARP ensures invariances to natural input transformations (e.g., permutations of node ids, tunnel reordering), and has a neural architecture aligned to the optimization model. Evaluations on the AnonNet dataset show HARP achieves an MLU at most 11% higher than optimal in over 98% of topology snapshots despite encountering significantly different topologies in testing relative to training data. Experiments on larger scale topologies show HARP provides computation benefits over Gurobi while being better able to handle topology perturbations not seen in training data relative to existing ML-based TE schemes. Finally, when predicted traffic matrices are provided, HARP outperforms Gurobi achieving a median reduction in MLU of 5 to 10% on the true traffic matrix.

While promising, the results are a start. HARP has focused on the MLU metric, and future work must investigate its applicability to other metrics such as MaxFlow, and fairness related metrics. Further, while we have focused on topology variations, the ability to handle significant changes in demand distribution is another area that requires investigation. Finally, investigating and improving tail performance of ML-based TE schemes including HARP is another interesting area for further exploration.

This work does not raise any ethical issues.

Acknowledgement. We thank our shepherd, Francis Yan, and the anonymous reviewers for their feedback which helped to improve the paper. This work was funded in part by the National Science Foundation (NSF) awards, CCF-1918483, CAREER IIS-1943364, CNS-1910234, and CNS-2212160, Amazon Research Award, AnalytiXIN, and the Wabash Heartland Innovation Network (WHIN), Ford, NVidia, CISCO, Amazon, and AFOSR FA95502210069.

[3] Code available at https://github.com/Purdue-ISL/HARP/

REFERENCES

[1] Abilene traffic matrices. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/.

[2] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. Contracting wide-area network topologies to solve flow problems quickly. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 175–200. USENIX Association, April 2021.

[3] David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proceedings of ACM SIGCOMM*, pages 313–324, 2003.

[4] David L. Applegate, Mateo D'iaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *Neural Information Processing Systems*, 2021.

[5] Beatrice Bevilacqua, Kyriacos Nikiforou, Borja Ibarz, Ioana Bica, Michela Paganini, Charles Blundell, Jovana Mitrovic, and Petar Veličković. Neural algorithmic reasoning with causal regularisation. *ICML*, 2023.

[6] Beatrice Bevilacqua, Yangze Zhou, and Bruno Ribeiro. Size-invariant graph representations for graph classification extrapolations. In *International Conference on Machine Learning*. PMLR, 2021.

[7] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjorner, Asaf Valadarsky, and Michael Schapira. Teavar: Striking the right utilization-availability balance in wan traffic engineering. In *Proceedings of ACM SIGCOMM*, 2019.

[8] Yiyang Chang, Chuan Jiang, Ashish Chandra, Sanjay Rao, and Mohit Tawarmalani. Lancet: Better network resilience by designing for pruned failure sets. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3:1–26, 12 2019.

[9] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

[10] E. Danna, S. Mandal, and A. Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *Proceedings of IEEE INFOCOM*, pages 846–854, 2012.

[11] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

[12] Google Developers. Choice of solvers and algorithms. https://developers.google.com/optimization/lp/lp_advanced#choice_of_solvers_and_algorithms/, 2024.

[13] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. https://github.com/pyg-team/pytorch_geometric, 05 2019.

[14] Bernard Fortz and Mikkel Thorup. Robust optimization of OSPF/IS-IS weights. In *Proceedings of International Network Optimization Conference*, pages 225–230, 2003.

[15] Nan Geng, Mingwei Xu, Yuan Yang, Chenyi Liu, Jiahai Yang, Qi Li, and Shize Zhang. Distributed and adaptive traffic engineering with deep reinforcement learning. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10, 2021.

[16] A. Ghosh, Sangtae Ha, E. Crabbe, and J. Rexford. Scalable multi-class traffic management in data center backbone networks. *IEEE Journal on Selected Areas in Communications*, 31:2673–2684, 2013.

[17] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proceedings of ACM SIGCOMM*, pages 350–361, 2011.

[18] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. Evolve or die: High-availability design principles drawn from googles network infrastructure. In *Proceedings of ACM SIGCOMM*, pages 58–72, 2016.

[19] Michael Gschwind, Driss Guessous, and Christian Puhrsch. Accelerated pytorch 2 transformers. https://pytorch.org/blog/accelerated-pytorch-2/, March 2023.

[20] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[21] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[22] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of ACM SIGCOMM*, pages 15–26, 2013.

[25] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, 2018.

[26] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined wan. In *Proceedings of ACM SIGCOMM*, pages 3–14, 2013.

[27] Chuan Jiang, Zixuan Li, Sanjay Rao, and Mohit Tawarmalani. Flexile: Meeting bandwidth objectives almost always. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, page 110–125, New York, NY, USA, 2022. Association for Computing Machinery.

[28] Chuan Jiang, Sanjay Rao, and Mohit Tawarmalani. Pcf: Provably resilient flexible routing. In *Proceedings of ACM SIGCOMM*, page 139–153, 2020.

[29] Grigorios Kakkavas, Michail Kalntis, Vasileios Karyotis, and Symeon Papavassiliou. Future network traffic matrix synthesis and estimation based on deep generative models. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–8, 2021.

[30] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: responsive yet stable traffic engineering. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, page 253–264, New York, NY, USA, 2005. Association for Computing Machinery.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

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

[33] Simon Knight, Hung Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29:1765 – 1775, October 2011.

[34] Alok Kumar, Sushant Jain, Uday Naik, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Proceedings of ACM SIGCOMM*, 2015.

[35] 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 *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, 2018.

[36] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019.

[37] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. In *Proceedings of ACM SIGCOMM*, pages 527–538, 2014.

[38] Libin Liu, Li Chen, Hong Xu, and Hua Shao. Automated traffic engineering in sdwan: Beyond reinforcement learning. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 430–435, 2020.

[39] T. Mallick, M. Kiran, B. Mohammed, and P. Balaprakash. Dynamic graph neural network for traffic forecasting in wide area networks. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1–10, Los Alamitos, CA, USA, dec 2020. IEEE Computer Society.

[40] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. Characterization of failures in an operational ip backbone network. *IEEE/ACM Trans. Netw.*, 16(4):749–762, 2008.

[41] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[42] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[43] Pooria Namyar, Behnaz Arzani, Srikanth Kandula, Santiago Segarra, Daniel Crankshaw, Umesh Krishnaswamy, Ramesh Govindan, and Himanshu Raj. Solving Max-Min fair resource allocations quickly on large graphs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1937–1958, Santa Clara, CA, April 2024. USENIX Association.

[44] Laisen Nie, Dingde Jiang, Lei Guo, and Shui Yu. Traffic matrix prediction and estimation based on deep learning in large-scale ip backbone networks. *Journal of Network and Computer Applications*, 76:16–22, 2016.

[45] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007. http://sndlib.zib.de, extended version accepted in Networks, 2009.

[46] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. *Networks*, 55(3):276–286, 2010.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[48] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. DOTE: Rethinking (predictive) WAN traffic engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1557–1581, Boston, MA, April 2023. USENIX Association.

[49] Michal Pióro and Deepankar Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[50] Rahul Potharaju and Navendu Jain. When the network crumbles: An empirical study of cloud network failures and their impact on services. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 15:1–15:17, 2013.

[51] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, page 140–151, New York, NY, USA, 2019. Association for Computing Machinery.

[52] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[53] Daniel Turner, Kirill Levchenko, Alex C. Snoeren, and Stefan Savage. California fault lines: Understanding the causes and impact of network failures. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 315–326, 2010.

[54] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *SIGCOMM Comput. Commun. Rev.*, 36(1):83–86, jan 2006.

[55] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets '17, page 185–191, New York, NY, USA, 2017. Association for Computing Machinery.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[57] Petar Velivckovi'c, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Mikhail Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, 2022.

[58] Ye Wang, Hao Wang, Ajay Mahimkar, Richard Alimi, Yin Zhang, Lili Qiu, and Yang Richard Yang. R3: Resilient routing reconfiguration. In *Proceedings of ACM SIGCOMM*, pages 291–302, 2010.

[59] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

[60] Shenghe Xu, Murali Kodialam, T. V. Lakshman, and Shivendra S. Panwar. Learning based methods for traffic matrix estimation from link measurements. *IEEE Open Journal of the Communications Society*, 2:488–499, 2021.

[61] Zhiying Xu, Francis Y. Yan, Rachee Singh, Justin T. Chiu, Alexander M. Rush, and Minlan Yu. Teal: Learning-accelerated optimization of wan traffic engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 378–393, New York, NY, USA, 2023. Association for Computing Machinery.

[62] 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 *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, page 1871–1879. IEEE Press, 2018.

[63] Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, and H. Jonathan Chao. Cfr-rl: Traffic engineering with reinforcement learning in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2249–2259, 2020.

[64] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 258–271, New York, NY, USA, 2021. Association for Computing Machinery.

T	Total number of tunnels per flow Assumes T is same for all tunnels.
d_{st}	Demand from s to t
γ_{stk}	k^{th} tunnel from s to t
u_{stk}	Unnormalized traffic on k^{th} tunnel from s to t
w_{stk}	The fraction of traffic from s to t that goes on its k^{th} tunnel. $\sum_{k=1}^{T} w_{stk} = 1$
l_{stk}	Bottleneck link of the k^{th} tunnel from s to t
$U(e)$	Utilization of link e
MLU	MLU of the network (i.e. utilization of most congested link in entire network)

Table 2: Notation Table.

A APPENDIX A

Appendices are supporting material that has not been peer-reviewed.

A.1 HARP implementation details

Figure 14 depicts our implementation for the GNN in Figure 2. We start by feeding the network topology and capacities to L GCN-Conv [32] layers where the node features are: i) summation of the capacities of all links connected to that node ii) degree of that node. After L GNN layers, the node embeddings of node i, NE_i are the concatenation of the node embeddings generated by every single layer. Then, the edge embeddings of edge$_{ij}$, h_{ij} are as described in Section 3.3. For the next modules, the implementation uses standard layers for SETTRANS (TransformerEncoder) and MLPs as implemented by PyTorch[47]. However, PyTorch advanced/sophisticated indexing is needed to capture the inputs required by MLP2. Using PyTorch indexing is necessary to avoid for-loops which will significantly impact the speed of HARP.

A.2 Hyperparameter Search

As mentioned before in Section 4, we did a grid search for every set of hyperparameters. For each hyperparameter setting, (i) we trained for sufficient epochs until the systems converged or stopped; and (ii) saved the model after every epoch. Given N combinations of hyperparameters and M epochs, this resulted in upto NM models totally of which we picked the best on the validation set.

DOTE. For DOTE, the hyperparameters (and the corresponding values) that we searched by default for all datasets included: (i) learning rate (1e-3, 3e-3, 5e-3); and (ii) batch size (32,256). For AnonNet, we found the performance greatly improved with even lower learning rates, and we additionally searched the following values (2e-5, 5e-5, 5e-4).

For TEAL, the hyperparameters (and corresponding values) searched were (i) learning rate (1e-4, 5e-4, 1e-3); (ii) batch size (32, 256); (iii) the number of samples to estimate the reward (5, 10); (iv) the number of FlowGNN layers (6, 8); and (v) multiple pseudorandom number generator seeds as recommended in training DeepRL models [9, 21]. In addition, we tried additional values beyond those listed above for each hyperparameter using a random search. We also found in our experiments that the performance of TEAL was sensitive to the units in which the traffic matrix and link capacities

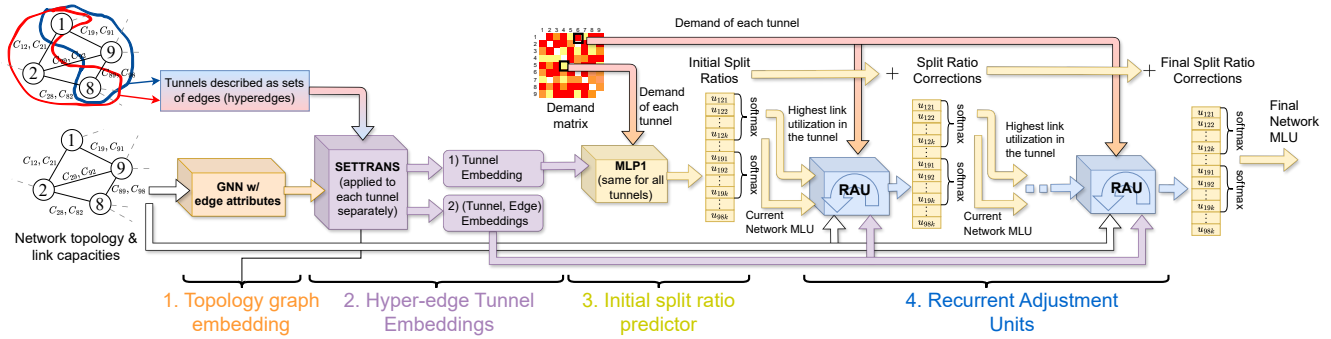


Figure 13: Simplified schematic diagram of HARP

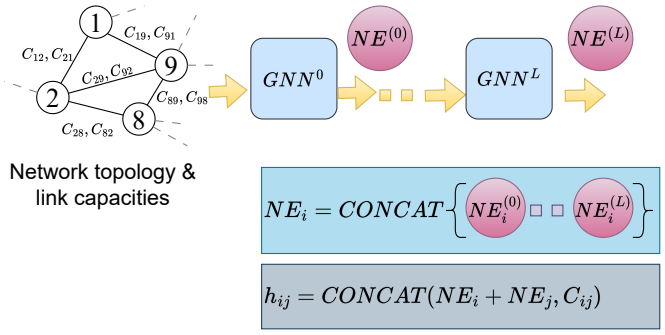


Figure 14: GNN Implementation of HARP

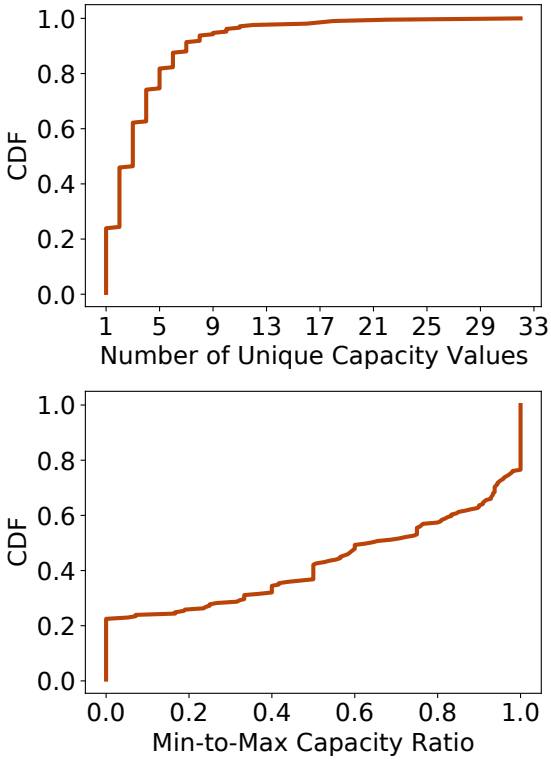


Figure 15: Variation of link capacities over entire AnonNet dataset. See 5.1 for discussion.

were supplied. For each dataset, we renormalized matrix values and capacity entries, tried multiple renormalization factors, and reported the best.

For HARP, the hyperparameters (and corresponding values) searched were (i) number of GNN layers (2, 3, 6); (ii) number of SETTRANS layers (2, 3); (iii) number of RAU units (3, 7, 14); (iv) learning rate (1e-3, 2e-3, 4e-3, 7e-3); (v) batch size (32, 256).

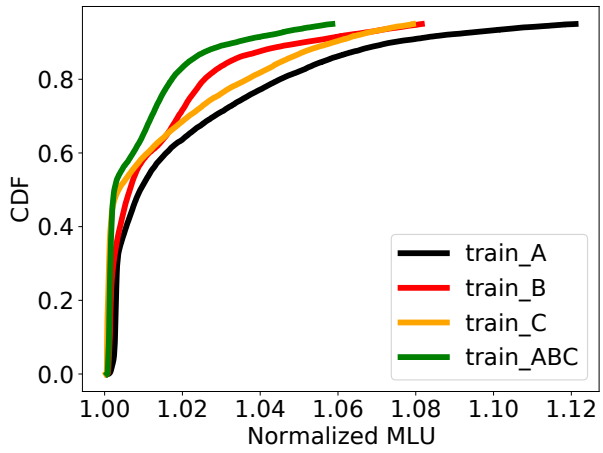
A.3 HARP's transferability

We expand on our discussion in §5.2. Using the data from the first three clusters (Clusters A, B, and C), we trained four different models (i) train_A; (ii) train_B; (iii) train_C; and (iv) train_ABC. The first three models were trained and validated using snapshots from a single cluster (A or B or C), while the fourth model was trained on snapshots of the three clusters cumulatively (and validated on three other clusters). We tested all four models on over 20K snapshots belonging to the remaining 72 clusters which were outside the training and validation sets of all four models.

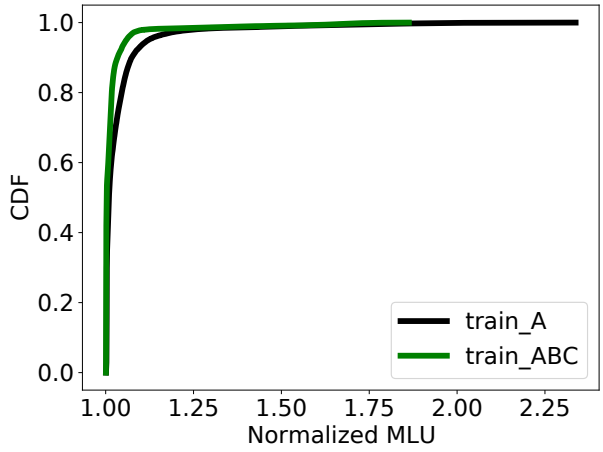
Figure 16 (top) presents a CDF of the NormMLU achieved by HARP across the snapshots in the testing set for each of the four models above. The graph is truncated at the 95%ile for clarity. While all models perform relatively well, the 95%ile for train_ABC is only 1.058, while the corresponding value for train_A (the worst performing of the three models trained on a single cluster) is slightly worse (1.12). Figure 16 (bottom) shows the full CDF of train_ABC and train_A (the other curves are omitted for clarity). Clearly, train_ABC improves the tail. The maximum is reduced to 1.86 from 2.33. The results overall show that HARP can generalize to other clusters even when trained with data from a very small number of clusters, and also indicate that its tail performance can be further improved by training on more clusters.

A.4 TEAL Convergence

In the AnonNet dataset we do not report results for TEAL as we were unable to get the training process to converge despite extensive hyperparameter search as described above. Figure 18(b) illustrates this by presenting the performance of TEAL during the

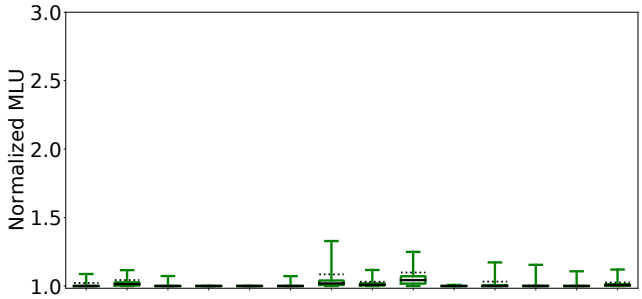


(a)

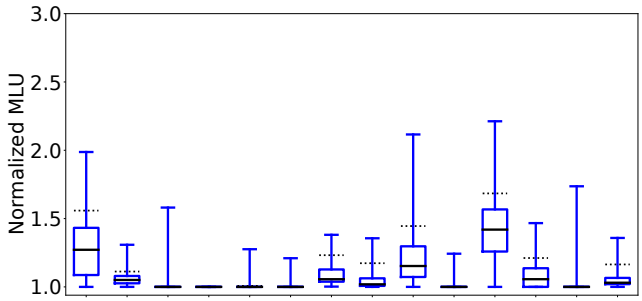


(b)

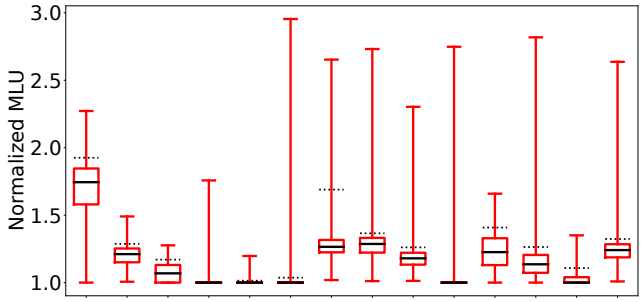
Figure 16: Comparing HARP's performance when trained on three clusters with its performance when trained on a single cluster. Top plot shows up to 95th percentile, bottom plot shows the entire CDF. Only results from the worst of three models trained on a single cluster each are shown in the bottom graph for clarity.



(a) HARP.



(b) DOTE.

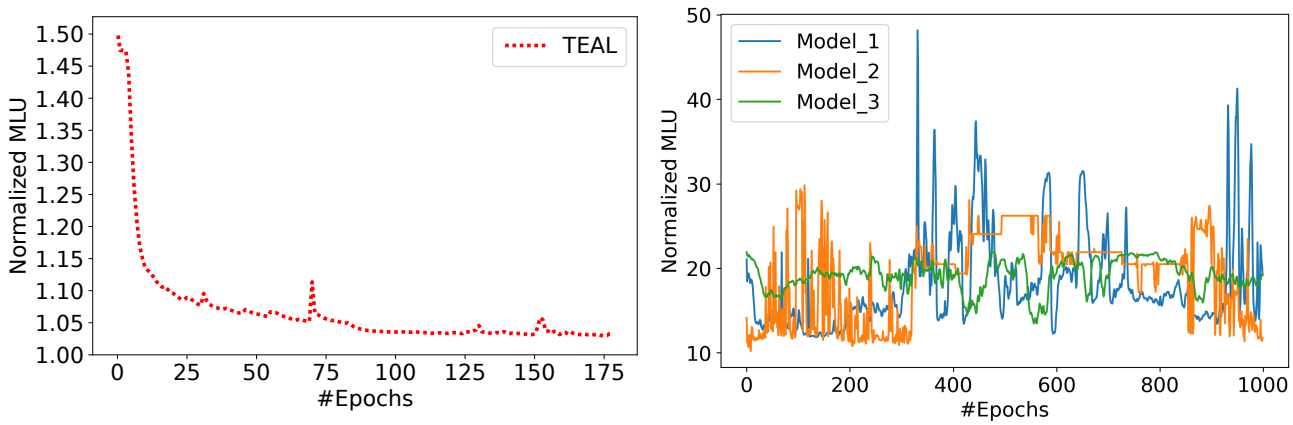


(c) TEAL.

Figure 17: NormMLU of HARP, DOTE and TEAL under different link failure scenarios for Abilene. The top whisker goes up to the maximum, while dashed lines show the 90th percentile.

training process for multiple example models (corresponding to different hyperparameter settings) including the one that performed the best on validation data. The results are shown for training on examples of the same AnonNet cluster (note that within a cluster there is significant variation in capacities of links owing to both partial and full failures as discussed earlier). The X-Axis corresponds to the training epoch, and the Y-Axis shows the the median NormMLU of the entire training set achieved during that epoch. Note that each epoch involves multiple mini-batches, and the MLU for training examples in subsequent mini-batches of an epoch may benefit by model updates made in prior mini-batches of that epoch. In all cases, the median of NormMLU remains high indicating the training process does not converge. While the models above are shown for illustrative purposes, we observed similar trends for training with other hyperparmeter settings as well.

Figure 18(a) presents training convergence results with TEAL for the KDL topology where the link capacities remain the same across all training examples. In clear contrast to AnonNet, the training process clearly converges for KDL. Note that the results in [61] are reported for settings with static link capacities, and our results for these settings are consistent with [61]. Investigating the convergence of the training process with TEAL in settings such as AnonNet which exhibits significant variability in link capacity (not



(a) KDL

(b) AnonNet

Figure 18: TEAL's learning curves on AnonNet and KDL topologies

the primary target of TEAL) requires further investigation which we defer to future work.

More generally, it is not uncommon to have this behavior of convergence for some datasets and not for others with methods that, like TEAL, use deep reinforcement learning [22, 48]. Addressing questions around training sensitivity with DeepRL techniques is an active area of research in the ML community [22].