Network Planning with Deep Reinforcement Learning

Hang Zhu
Johns Hopkins University

Varun Gupta
Facebook Inc.

Satyajeet Singh Ahuja
Facebook Inc.

Yuandong Tian
Facebook Inc.

Ying Zhang
Facebook Inc.

Xin Jin
Peking University

ABSTRACT

Network planning is critical to the performance, reliability and cost of web services. This problem is typically formulated as an Integer Linear Programming (ILP) problem. Today's practice relies on hand-tuned heuristics from human experts to address the scalability challenge of ILP solvers.

In this paper, we propose NeuroPlan, a deep reinforcement learning (RL) approach to solve the network planning problem. This problem involves multi-step decision making and cost minimization, which can be naturally cast as a deep RL problem. We develop two important domain-specific techniques. First, we use a graph neural network (GNN) and a novel domain-specific node-link transformation for state encoding, in order to handle the dynamic nature of the evolving network topology during planning decision making. Second, we leverage a two-stage hybrid approach that first uses deep RL to prune the search space and then uses an ILP solver to find the optimal solution. This approach resembles today's practice, but avoids human experts with an RL agent in the first stage. Evaluation on real topologies and setups from large production networks demonstrates that NeuroPlan scales to large topologies beyond the capability of ILP solvers, and reduces the cost by up to 17% compared to hand-tuned heuristics.

CCS CONCEPTS

• **Networks → Network management**; • **Theory of computation → Reinforcement learning**;

KEYWORDS

Network planning, Reinforcement learning, Graph neural network

ACM Reference Format:
Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. 2021. Network Planning with Deep Reinforcement Learning. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, Netherlands.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3452296.3472902

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472902

1 INTRODUCTION

Large-scale web services rely on high-performance, reliable Wide Area Networks (WANs). From cloud computing and big data, to Artificial Intelligent (AI) and Internet of Things (IoT), the demand for WAN bandwidth keeps growing rapidly under these technology trends [2]. Network planning is a regular, critical process to plan and upgrade WANs, in order to meet the performance and reliability requirements of web services while minimizing the cost.

Network planning is a hard combinatorial optimization problem. Cross-layer decisions that involve both the IP layer and the optical layer need to be made. The planned network must satisfy certain service expectations specified by the operator, which include both performance requirements (e.g., sufficient bandwidth for given traffic matrices) and reliability requirements (e.g., robust to failures). The cross-layer nature of the problem makes it particularly challenging as a failure in the optical layer may affect multiple links in the IP layer. Given the high cost of building WANs, network planning must minimize the network cost while satisfying the service expectation requirements.

To solve this problem, we can formulate network planning as an Integer Linear Programming (ILP) problem. The constraints encode the service expectations and the objective is to minimize the cost. We can use an off-the-shelf ILP solver, such as Gurobi [19] and CPLEX [11], to find the optimal solution that satisfies the service expectations and has the minimum cost.

The key problem of the naive ILP approach is scalability. It does not scale to large topologies beyond tens of nodes. In practice, operators integrate *hand-tuned* heuristics from human experts into the process, e.g., adding human-designed constraints to limit the search space of the ILP solver. These heuristics need to make a *trade-off* between the optimality of the solution and the tractability of the problem. Further, even highly-skilled experts require many iterations to find such a good trade-off manually. Accounting for the time to run ILP for each iteration, this approach not only requires extensive human expertise and involvement, but is also time-consuming. Moreover, there are no universal heuristics that can achieve a good trade-off for all networking planning scenarios. Operators have to manually examine and repeat the iterative process for every scenario.

In this paper, we propose NeuroPlan, a deep reinforcement learning (RL) approach to solve the network planning problem. There are two characteristics of this problem that make it a good fit for RL. First, network planning needs to make multiple decisions on the IP and optical layers, and multi-step decision making is exactly the kind of problems that RL is designed to solve. Recent results have demonstrated the capability of RL in solving hard combinatorial problems that cannot be easily solved by conventional solutions [28, 32, 45]. Second, the effects of the decisions (e.g., deciding the capacity of one IP link) can only be evaluated once the entire network has been planned. Hand-tuned heuristics often focus on local decisions to maximize (implicit) local metrics, while RL is designed to handle delayed rewards and explicitly optimize for the global objective.

While many problems in different domains have these characteristics, it is particularly appealing to apply deep RL to network planning. Deep RL is notorious for its high sample complexity. Many domains rely on simulations to obtain enough samples for training. Yet, there is a gap between simulation and practice (e.g., in robotics), and this is a major obstacle for deep RL solutions to be used. In network planning, however, there is no such gap. A network plan generated by an RL agent can be accurately and efficiently evaluated by calculating its cost and checking the service expectation requirements at the cost of cheap CPU cycles in the same way as a plan evaluator is used now in production by network operators. And many network plans can be sampled to train the RL agent.

Given the potential of this deep RL approach, we need to address two technical challenges to realize its promise. First, we need to represent the network topology with various numbers of nodes and links into a state vector for the RL agent to use, and the representation needs to handle topology dynamics when the RL agent applies actions to change the topology (e.g., adding IP capacity). We address this problem by encoding the network topology with a graph neural network (GNN) and learn an embedding vector representation. Moreover, we design a novel domain-specific node-link transformation to transform the topology before feeding it to the GNN. This transformation is critical because network planning is primarily concerned with the bandwidth provided by links and directly using GNNs cannot handle parallel links.

Second, deep RL is not a panacea. One straightforward approach is to directly use it to generate the final network plan. Deep RL can learn to find reasonably good solutions relatively quickly, but given the combinatorial nature of the problem, converging to the optimal solution is fundamentally hard. Instead, we leverage a hybrid approach which contains two stages. The first stage uses deep RL to learn to prune the search space, and the second stage uses an ILP solver to find the optimal solution. Because the search space has been pruned significantly by deep RL, the ILP solver can finish the second stage quickly. This approach resembles today's practice of using hand-tuned heuristics to prune the search space. Remarkably, our approach does so without the need of human experts—*the RL agent replaces human experts to automatically generate pruning strategies.*

Our approach is incrementally deployable and interpretable. Network operators can examine the pruning strategies generated by deep RL and check whether they match their intuition and experience for interpretability. NeuroPlan is not intrusive to the current practice of network planning. Network operators can decide whether to incorporate the pruning strategies generated from RL together with their hand-designed strategies. Alternatively, they can keep using their existing tools to generate network plans, and compare these plans with that generated by NeuroPlan to decide which plan to use. In addition, NeuroPlan also provides a knob for operators to easily and explicitly tune between optimality and running time of the ILP solver. We are in the process of incorporating this solution to the operational planning process in a large WAN.

In summary, we make the following contributions.

- We propose NeuroPlan, a deep RL approach to solve the network planning problem.
- We design domain-specific techniques based on GNNs and node-link transformation for state encoding, and leverage a hybrid approach to find the optimal solution.
- We implement a NeuroPlan prototype. Evaluation on real topologies and setups from production networks demonstrates that NeuroPlan scales to large topologies beyond the capability of ILP solvers, and reduces the cost by up to 17% compared to hand-tuned heuristics. The 17% cost saving is significant given the high cost (e.g., billions of dollars) of building WANs and the tremendous efforts of developing and manually tuning heuristics for the baseline, and NeuroPlan relieves humans from these efforts.

Finally, there is an emerging trend towards self-driving networks [24, 34, 38, 39]. The over-arching goal is to bring automation to network management—operators only specify high-level intents and networks manage themselves. While the goal is tremendously attractive, it is an open question how to achieve it. In this paper, we make a concrete step towards this goal by bringing AI techniques and automation to an important network management task—network planning. Notably, our solution is not intrusive. It demonstrates the viability of self-driving networks and the potential of using AI techniques to relieve the burden of network operators.

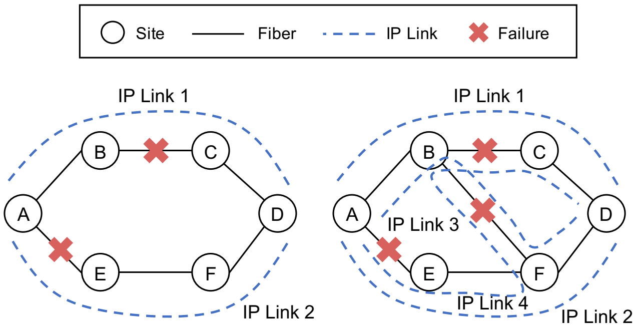
Open-source. The code of NeuroPlan is open-source and is publicly available at `https://github.com/netx-repo/neuroplan`.

2 NETWORK PLANNING PRIMER

Large content service providers operate a global network of tens of datacenters and hundreds of Point of Presence (PoP) sites. The backbone network interconnecting the datacenters and PoPs typically serves hundreds of terabits of traffic at any given time. Continuous changes in user demands, services, and traffic patterns dictate that the backbone network needs to continuously evolve to meet these needs. Importantly, operational issues such as upgrading networking hardware, scheduled maintenance, and failures also need to be taken into consideration to ensure high network reliability and performance. The network planning process is a critical step in dimensioning the backbone network to satisfy these requirements.

The high-level goal of network planning is to determine how to scale the network to satisfy the traffic given a future demand forecast. By working with a content provider's network planning team closely, we summarize a few properties of a real-world network planning process below.

First, it is *cross-layer* as it considers both the optical layer and the IP layer. At the IP layer, it determines the amount of capacity in Gbps for each IP link, as well as the number of transponders



(a) Short-term planning. (b) Long-term planning.

Figure 1: An example for network planning to satisfy a 100Gbps flow from A to D under any of the three single-fiber failures. (a) Short-term planning uses two IP links A-B-C-D and A-E-F-D. (b) Long-term planning adds a new fiber B-F, and uses two IP links A-B-C-D and A-B-F-D. Other options (e.g., using A-E-F-D and A-E-F-B-C-D) consume more fibers.

and routers to procure at each site to support such capacity. At the optical layer, it estimates the number of fibers to turn on, as well as new fiber path to build or purchase. Across layers, it layouts the mapping of IP links and underlying fiber paths. All of these decisions form a network plan that is being procured and deployed over the months and years. It contributes to the most significant portion of network infrastructure expenditure.

Secondly, network planning is a *multi-phased, iterative process.* Making all these decisions in one shot is complex and does not meet the operational needs. For instance, building new fiber path takes much longer time and thus such decision needs to be made several years a-priori. Thus, in production networks, the planning process is done over the short-term and long-term horizons respectively to generate short-term actionable signals for operational teams, while allowing optimal long-term network evolution strategy. Short-term planning provides decisions for adding or removing capacity over existing IP links on a given fiber footprint, for a time range of the next few months. Long-term planning, on the other hand, decides the future fiber paths, hardware equipments, and new sites. Note that the IP topology is up for change in this step. The goal is to extract the most cost-effective IP topology and the corresponding IP and optical equipment required to support future IP topologies.

Lastly, network planning is *failure-aware.* A key objective of network planning is to ensure the reliability of the backbone network under different failure scenarios such as fiber cuts, site failures and natural disasters. We illustrate the failure-aware multi-phase planning in an example in Figure 1. For simplicity, assuming the traffic demand is a single flow of 100Gbps from site A to site D, the goal is to plan a network of IP links that satisfies the demand and minimizes the cost, which is approximated as the number of fibers used. The failure scenarios to consider are marked as the red cross, each of which can happen independently. Short-term planning is to decide the capacity of the two existing IP links, shown as dotted lines. A single IP link (link 1: A-B-C-D or link 2: A-E-F-D) is not sufficient, because it would not survive the failures. As shown in Figure 1(a), we need to build both 100Gbps IP links to ensure that the flow from A to D is always satisfied when either A-E or B-C fails.

A long-term planning example is shown in Figure 1(b). Here we can change both the optical and IP topology. Assume that we decide to add a new fiber B-F, which also introduces another new possible failure. There are two options to add a new IP link between A and D, link 3 (A-B-F-D) and link 4 (A-E-F-B-C-D). Consider three possible plans: (1, 2), (1, 3), (2, 4), which can satisfy the demand under any of three possible failures. Plan (1,3) has a lower cost than the other two, because IP link 1 and 3 share one fiber A-B so that this plan only consumes 5 fibers while other plans use more fibers. This example illustrates that both long-term planning and failure scenarios would result in a potentially large search space to explore.

3 EXISTING APPROACH AND CHALLENGES

In this section, we describe a standard solution to network planning that is used in a production network. Importantly, we highlight the scalability challenge of the existing solution and summarize a few heuristic methods used to tackle the challenge. This challenge and the ad-hoc workarounds motivate us to explore more intelligent and more systematic solutions in the rest of this paper.

3.1 Problem Formulation

Symbol	Description
F	Fiber set
L	IP link set
N	IP node set
Λ	Failure set
Ω	Flow set
S_f	Maximum available spectrum over fiber f
ϕ_{lf}	Spectrum efficiency of IP link l over fiber f
C_l	Capacity (in Gbps) of IP link l
C_l^{min}	Minimum capacity (in Gbps) of IP link l
Δ_f	Set of IP links traversing over fiber f
Ψ_l	Set of fibers traversed by IP link l
$Y(l, \omega, \lambda)$	Traffic of flow ω over IP link l under failure λ
$cost_f$	Cost of building fiber f
$cost_{IP}$	Cost of turning up IP capacity per Km per Gbps

Table 1: Key notations in problem formulation.

At a high level, the backbone network topology is abstracted as a graph where each IP/optical site is represented as a node and the optical fibers between them make up the edges in a layer 1 (L1) topology. The IP links traversing over the fibers create an overlay IP topology, usually referred to as layer 3 (L3) topology. The backbone traffic is represented as flows between different sites with various Classes of Services (CoS) based on service characteristics. The overall goal of network planning is to minimize the total network cost while meeting all constraints imposed by hardware and operational requirements. The problem can be formulated as an optimization problem under a set of constraints. The mathematical formulation is described below. Table 1 summarizes the key notations of the problem formulation.

Objective. The objective of the ILP is to minimize the total network cost, which is a sum of the cost on optical layer and IP layer. Other metrics such as flow latency can also be included in the objective based on operational requirements. The cost of optical layer consists of one-time procurement cost of fiber pairs, one-time fiber light up and monthly operating cost. The cost of IP layer comes from buying IP equipments and operational cost to turn up IP capacities. For the sake of simplicity, we abstract the cost as the sum of IP link costs where each link's cost is proportional to the capacity added over the IP link as well as the the fiber cost underneath, shown as Eq 1.

$$\min \sum_{l \in L} (C_l \times cost_{IP} + \sum_{f \in \Psi_l} cost_f) \tag{1}$$

$$\text{s.t.} \sum_{l:l_{src}=n} Y(l,\omega,\lambda) - \sum_{l:l_{dst}=n} Y(l,\omega,\lambda) = Traffic(\omega,n)$$

$$\forall \omega \in \Omega, \lambda \in \Lambda \tag{2}$$

$$C_l \geq \sum_{\omega} Y(l,\omega,\lambda),\ \forall \lambda \in \Lambda \tag{3}$$

$$\sum_{l \in \Delta_f} C_l \times \phi_{lf} \leq S_f \tag{4}$$

$$C_l \geq C_l^{min} \tag{5}$$

The constraints in Equation 2-5 are explained below.

- *Flow conservation constraint (Eq. 2):* The amount of egress traffic needs to be equal to the amount of ingress traffic plus the node's self-generated traffic for every flow under every failure scenario. $Traffic(\omega,n)$ is: the volume of ω if n is the source of ω, negative of the flow volume if n is the sink, and 0 otherwise.

- *Link capacity constraint (Eq. 3):* The capacity of each IP link is dictated by the capacity requirement of the aggregated traffic volume on the link under any failure λ. Note that due to operational constraints, each IP link can only be turned up in fixed capacity unit, i.e., C_ls are integer variables in the formulation.

- *Spectrum consumption constraint (Eq. 4):* At the optical layer, all constraints are expressed as that the total spectrum consumed is less than the maximum available spectrum for each fiber [26, 65, 66, 68]. The spectrum consumed over a fiber is the sum of IP link capacities going over the fiber multiplied by the corresponding spectrum efficiency (i.e., spectrum consumption to support each IP capacity unit).

- *Existing topology constraint (Eq. 5):* For short-term planning, there is an additional constraint that each link's capacity should not deviate from existing production capacity by too much, denoted by C_l^{min}. This is to avoid churns of the topology from an operational perspective. For long-term planning, C_l^{min} is set to 0 for the candidate links to be added to the topology.

For brevity, we focus on the fundamental variables and constraints. Different routing protocols and traffic engineering system requirements (e.g., MPLS tunneling selection [8], OSPF [14]) can be incorporated into the problem formulation in practice.

3.2 Pain Points in Today's Approach

The network planning problem is formulated as an ILP problem and can be directly solved with an off-the-shelf ILP solver such as Gurobi [19] and CPLEX [11]. The primary issue of the ILP approach is the scalability of the ILP solver. For example, one of our production network has about 100 nodes, 300 links and 500 failure scenarios. In short-term planning, it is translated into an ILP problem with 42 million variables and 5 million constraints. In long-term planning, the number of variables goes up to 400 million. Even worse, the topology size grows at a rate of 20% per year. In practice, we heavily use hand-tuned heuristics based on human expertise to overcome the scalability challenge. For the long-term planning problem, we can only get an actionable signal after applying heuristics and running the ILP solver for 3-4 days. Below, we describe the pain points of our existing approach from our experiences of planning for a large scale WAN.

Short-term planning. In short-term planning, the IP topology is given, and the task is to decide the capacity on given IP links. For such cases, building and solving an ILP with thousands of variables and constraints can be done in a few minutes using existing commercial solvers. However, since the amount of computation needed grows exponentially when the topology size increases, the ILP approach cannot be directly used for large topologies. In such cases, we rely on heuristics to solve the problem, some of which are described below.

- *Topology decomposition.* We decompose the topology into several smaller sub-topologies, and each sub-topology is solved with an ILP. The decomposition is usually done by segmenting the topology into geographical regions where each region is aligned to actual operational/management blocks of the production network and sizing inter-regional links (generally subsea or long-haul terrestrial links). The segmentation and stitching are done manually.

- *Topology transformation.* We transform the topology to reduce the problem search space or reduce the number of variables. The transformation can include actions like enlarging the capacity unit that can be added over some or all links, restricting capacity additions on fibers or IP links, or collapsing multiple nodes and links together. These transformations are determined manually and decided based on the level of fidelity required from planning signals.

- *Failure selection.* Instead of satisfying all failure scenarios jointly, we only select a subset of the failure scenarios for the ILP (i.e., reducing the number of constraints and variables). Based on the solution, we add more failure scenarios to the ILP until all scenarios are added. The ordering that the failure scenarios are added is decided manually.

Different heuristics share the same idea—they all aim to *prune* the search space to make the problem tractable. Yet, they all rely on human expertise and operational experience. It is a tedious, time-consuming iterative process even for experts as the experts need to use trial-and-error for the manual parts in the heuristics. As the search space is pruned, there is a fundamental tradeoff between the optimality of the solution and the tractability of the ILP. This adds additional complexity to the problem, and yet another iterative process is employed to find a good trade-off. Even worse, such process has to be done and tuned for every planning problem as there are no universal heuristics for all the planning problems.

Long-term planning. The ILP approach hits the limit of scalability for even smaller topologies for long-term planning. As an example, the candidate set of possible IP links for the global topology at our scale is larger than ten thousand. Of these ten thousand candidates, fewer than five hundred may make it to the production topology based on ILP results. Another challenge is that most ILP solvers struggle to find an optimal integer solution when starting from almost zero IP capacities on the candidate links, compared to the short-term planning (see explanation of Eq. 5).

Most ILP solvers rely on estimating feasible solutions using proprietary heuristic methods and then using branch-and-cut techniques [44, 50] to eliminate sub-optimal solutions. Since the search space is large, the generic heuristic methods of solvers suffer in finding good quality solutions for these problems. Further, the possible number of branches for the long-term planning problem is too large. Besides the strategies used in short-term planning above, we leverage warm-start to feed potential feasible solutions to ILP solvers and help solvers converge faster to an optimal. Warm-start solutions can include previously known good designs or solutions from manually crafted heuristics. The warm-start solutions require considerable manual intervention and domain expertise, and are iterative and time-consuming.

4 NEUROPLAN DESIGN

4.1 A Deep RL Approach

The network planning problem is a multi-step decision making problem, which is exactly the set of problems that RL is designed to solve. Specifically, RL considers the problems where there is an agent interacting with an environment [46, 47]. The agent observes the environment, takes actions to change the environment, and receives rewards from the environment. Through a series of interactions, the agent learns a policy to maximize its rewards. The RL setup can be naturally mapped to the network planning problem. The environment is the network topology with the traffic demand and reliability policy, the actions are to change the link capacity of the topology, and the rewards are to indicate the final cost of the network topology and whether the traffic demand is satisfied under the reliability policy.

Advantage of deep RL approach. While many problems are fundamentally hard in terms of computational complexity, deep RL has been recently demonstrated to achieve remarkable results in many domains [1, 34, 43, 54, 59, 69, 74]. The key reason is that deep RL is able to leverage neural networks to learn the structure of the particular problem by exploring the search space and exploit the learned structure to optimize its policy. This is essentially how humans design heuristics to take advantage of the problem structure to solve the problem. The real appeal here is that the RL agent can automatically derive such heuristics using a deep RL algorithm, obviating the need to manually design and tune heuristics with human experts.

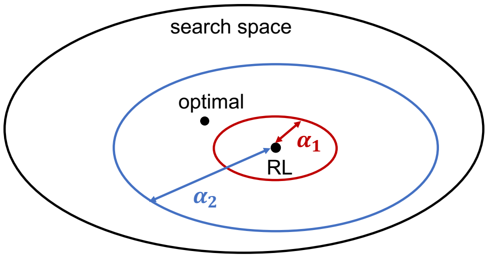


Figure 2: Two-stage hybrid approach in NeuroPlan. The first stage uses RL to find an initial solution, and the second stage uses ILP to find the final solution in a sub search space near the initial solution bounded by the relax factor α. The factor α provides a knob for the trade-off between optimality and tractability.

Moreover, in the language of RL, RL is able to model *delayed* rewards and directly optimize for the global objective. This is the reason that in many cases, deep RL algorithms can often beat human experts. With regard to our problem, the hand-designed heuristics like the ones we describe in §3.2 often focus on local decisions. For example, a heuristic to limit the capacity of an IP link to a small range (i.e., tighten the constraint) is often based on the flows on the IP link, not all flows and the entire topology. It is hard for human experts to model the relationship between the local decisions and the global objective in the heuristics, and as a result, the local decisions are often evaluated with local metrics that are only loosely coupled with the global objective. In comparison, deep RL can choose an action even if the impact of the action can only be evaluated after several steps, i.e., the reward of the action is delayed. Deep RL is able to learn to choose actions to directly optimize for the global objective.

An alternative is to apply supervised learning which uses a training set with labeled samples to train a neural network. However, supervised learning usually requires a large training set and a label for each sample (i.e., optimal plan for a network) in the training set, both of which are hard to obtain for our problem, especially for large-scale network topologies. The key benefit of deep RL is that it is able to explore the search space automatically and efficiently.

Challenge 1: Dynamic network topology. While it is natural and appealing to apply deep RL to network planning, there are two primary technical challenges. The first challenge is to model dynamic network topology in deep RL. A deep RL agent typically requires a vector to encode the environment state as the input of its neural network to generate actions. However, the environment of the network planning problem is the network topology, which is a graph. The size of a graph varies depending on the number of nodes and edges, and the structure cannot be easily captured with a vector. To make it more challenging, the graph is dynamic when the RL agent interacts with the topology to change the capacity of IP links.

To address this challenge, we leverage graph representation learning [13] to encode the network topology with GNNs [52]. GNNs are a family of neural networks specifically designed for graph data [78]. It takes a graph as input and can learn to generate

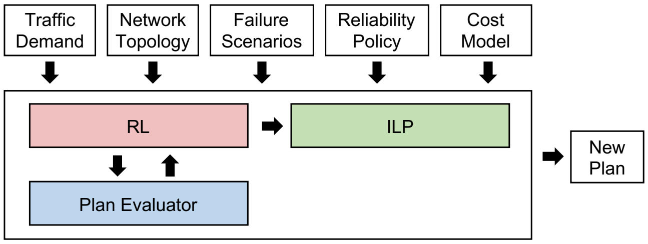


Figure 3: The workflow of NeuroPlan.

embedding vectors to represent its nodes, links or the entire graph. It can support many node-level, link-level or graph-level tasks. For our problem, GNNs are better than other sequence-based graph representations (e.g., recurrent neural networks [16]) since GNNs eliminate the dependency of the order where the nodes are given in the input. We take the dynamic network topology as the input of a GNN to generate an embedding vector for the entire topology. The embedding vector captures the essential features of the topology, and serves as the input of the RL agent. One characteristic about our problem is that we care about edges (i.e., the capacity of the IP links), not nodes, of the graph. And there exist parallel links between two IP nodes, which are mapped to different fiber paths and are associated with different failure scenarios. The novelty in our approach is that we use a domain-specific node-link transformation to transform the topology based on this characteristic to improve learning efficiency and support parallel links.

Challenge 2: Trade-off between optimality and tractability. A straightforward approach to apply deep RL to network planning is to directly use the RL agent to generate the network plan. Given the combinatorial nature of the problem, it is fundamentally hard for the RL agent to find the optimal solution. The RL agent can learn to find reasonably good solutions, but it is both unpredictable and time-consuming for the agent to find the optimal solution.

To address this challenge, we leverage a two-stage hybrid approach with RL and ILP. Instead of using RL to directly generate the final solution, we use it to prune the search space and bootstrap ILP. Figure 2 illustrates our approach. It is infeasible to search the entire space to find the optimal solution with ILP due to the large and complex search space. In the first stage, we use deep RL to learn to find a reasonable solution. In the second stage, we use ILP to only search the space near the solution found by deep RL. We use the relax factor α to control the size of the space to explore by ILP. The relax factor α provides a *tunable* knob for the network operator to trade-off between optimality and tractability. A large α (α_2 in Figure 2) allows ILP to explore larger space, but the problem may be intractable or takes a very long time for the ILP solver to solve. On the other hand, a small α (α_1 in Figure 2) may not include the optimal solution in the search space, but the ILP solver may finish the second stage quickly.

Workflow of NeuroPlan. Figure 3 shows the workflow of NeuroPlan. The input of NeuroPlan consists of five components, i.e., traffic demand, network topology, failure scenarios, reliability policy and cost model. The RL agent only needs to encode the network topologies. The other four components are handled by the plan evaluator. The RL agent interacts with the plan evaluator to learn to generate network plans that minimize the network cost while satisfying the traffic demand under the reliability policy. The plan evaluator generates rewards to the RL agent. It receives the network plan from the RL agent, checks whether the traffic demand is satisfied under different failure scenarios, and uses the cost model to compute a cost of the plan. The reliability policy specifies the demand of flows with which Classes of Service (CoS) has to be satisfied under which subset of failure scenarios. A reward is calculated based on a combination of whether the demand is satisfied and the cost of the plan. We describe the details of RL agent and the reward encoding in §4.2. After the learning process is completed, the RL agent outputs an initial plan for the first stage. At the second stage, the ILP solver uses ILP to find the final plan in the pruned search space near the initial plan defined by the relax factor α. We describe the details of pruning the search space using the initial plan and the relax factor α in §4.3.

Unifying short-term and long-term planning. We use NeuroPlan to unify short-term and long-term planning with the same approach. The main difference between short-term and long-term planning is whether the IP links are given. Our key observation is that the starting topology for long-term planning can be considered as a topology with all the candidate IP links with the starting capacity to be zero. These candidate IP links are mapped to different fiber paths that are currently not used but can potentially be procured or built to the topology. Then both short-term and long-term planning can be solved by the same agent that decides capacity to a given topology until the traffic demand is satisfied under the reliability policy.

4.2 NeuroPlan Training Algorithm

Figure 4 shows the process of the RL agent to generate a network plan from the original network topology. For each step i, the RL agent performs an action to update the network topology (e.g., add some capacity to an IP link). The plan evaluator checks the updated topology to see if the traffic demand is satisfied under the reliability policy. The condition for a trajectory to stop is that the traffic demand is satisfied under the reliability policy or the number of steps (i.e., the number of actions taken) is equal to a pre-defined threshold. It is a common practice for deep RL to set a maximum number of steps to stop a trajectory even if the task (e.g., find a plan that satisfies the traffic demand under the reliability policy) is not completed. This helps improve training efficiency as the RL agent is early stopped on unpromising trajectories. If the current trajectory is terminated, the RL agent starts a new trajectory from the original network topology to gain more experience to train its neural network. Otherwise, the agent performs another action for step i+1. For each step, the agent receives an award after it performs its action.

Node-link transformation. While there are a variety of GNNs for different graph data and tasks [1, 43, 78], GNNs are most mature for handling node features and performing node tasks such as node classification [29] and node property prediction [74]. In the context of network planning, however, we care about links, not nodes. Specifically, we would like the GNN to generate a high-level graph embedding based on the current capacity of the IP links (i.e.,

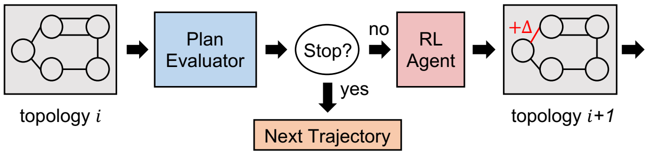
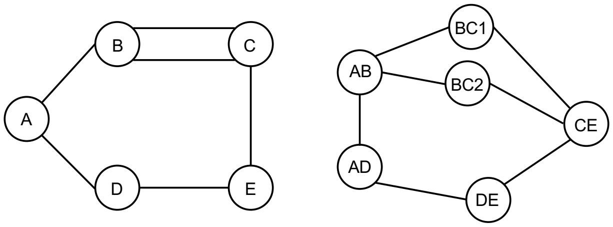


Figure 4: The process of RL training to generate a network plan.



(a) Before transformation. (b) After transformation.

Figure 5: Node-link transformation can handle parallel links. The node names show the corresponding relationship between nodes and links.

link features) and the RL agent can utilize the graph embedding to decide which IP links should have more capacity (i.e., link tasks). Moreover, there are parallel links (mapped to different fiber paths) between two nodes, making it harder for the GNN to encode the features as well as the graph structure.

We design a domain-specific node-link transformation to transform the input network topology before feeding the topology to the GNN. Unlike complement graph [6] which fills in all the missing edges to form a complete graph, or dual graph [48] that maps faces to vertexes, the main idea of the transformation is to map each link in the input topology to a node in the transformed topology. Then the link features (i.e., link capacity) in the input topology become the node features in the transformed topology. This makes it easier to apply state-of-art GNN algorithms to the topology. The nodes in the transformed topology are connected if their corresponding links have one common end in the input topology. Figure 5 provides an example to illustrate the node-link transformation. Figure 5(a) is the input topology before transformation. There are five nodes (A, B, C, D, E) and six links (AB, AD, DE, CE, BC1, BC2) in total. BC1 and BC2 are two parallel links between node B and node C. Figure 5(b) is the network topology after transformation. The two parallel links BC1 and BC2 are mapped to two nodes in the transformed topology. We do not add links between nodes whose corresponding links in the input topology are parallel links, e.g., we do not add links between node BC1 and node BC2 in Figure 5(b). This is because the parallel links make contribution to the capacity between the same two nodes, and we do not want to propagate the capacities of the parallel links during GNN training.

State representation. The state representation includes the network topology and the node features. After the node-link transformation, the links in the input topology become the nodes in the transformed topology. We use the current capacity of the IP link as the feature for each node in the transformed topology. For training efficiency, we normalize each dimension of the node feature across all the nodes in the transformed topology with $mean = 0, std = 1$. This is because an RL agent tends to generate the same action if the values are the same in most of the dimensions in the input state. Normalization is a commonly-used technique to avoid generating a consecutive sequence of same actions.

Algorithm 1 Learning a plan-generation policy using an actor-critic algorithm.

Input: The network topology with the original capacity $\mathbf{G}^*$.

Main routine:

```
// Initialization
Initialize the actor parameters $\theta$, the critic parameters $\theta_v$, the GNN parameters $\theta_g$
Number of epochs $N$
$G \leftarrow$ Reset($\mathbf{G}^*$)
for n=1,2,...,N do
    // Generate several network plans with the current actor
    $epochBuffer.clear()$
    while !$EpochEnd$ do
        $log\ p \leftarrow \pi(a|G; \theta, \theta_g)$
        $a \leftarrow log\ p.sample()$
        $v \leftarrow V(G; \theta_v, \theta_g)$
        $G, r \leftarrow$ UpdateTopo($G$, $a$)
        $epochBuffer.append(log\ p, a, v, r, G)$
        if $TrajEnd(G)$ then
            $G \leftarrow$ Reset($\mathbf{G}^*$)
    Reset gradients $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$ and $d\theta_g \leftarrow 0$
    // Compute gradients wrt. actor gradient loss
    $d\theta, d\theta_g \leftarrow ComputePLoss(epochBuffer)$
    Perform update of $\theta$ using $d_\theta$ and $\theta_g$ using $d\theta_g$
    // Compute gradients wrt. critic gradient loss
    $d\theta_v, d\theta_g \leftarrow ComputeVLoss(epochBuffer)$
    Perform update of $\theta_v$ using $d\theta_v$ and $\theta_g$ using $d\theta_g$
```

Subroutines:

- Reset(G): Reset the network topology G to its initial state.
- UpdateTopo(G, a): Apply action a to network topology G, and return the updated topology and the intermediate reward for the action.
- TrajEnd(G): Define the condition of the current trajectory end as G satisfies the service expectations or the length.
- ComputePLoss(epochBuffer): Compute the advantage estimates, and update the gradient as the mean error between it and $log\ p$.
- ComputeVLoss(epochBuffer): Compute the rewards-to-go, and update the gradient as the mean-square error between it and v.

Action representation. The action representation indicates which link to select to *add* capacity and how much capacity to add. Let the largest amount of capacity unit to add in one step be *m* and the number of nodes in the transformed topology is *n*. The size of the action space is *mn*.

An alternative approach is to allow both *adding* and *reducing* capacity for IP links in the actions. It is important to note that both approaches cover the same search space, i.e., any plan found by the alternative can be found by only allowing *adding* capacity in the actions. There are three benefits to only allow adding capacity in the actions. First, it leads to a smaller action space. The size of the action space with only adding capacity is half of that with both adding and reducing capacity. Second, it leads to a stable and simple training process. A trajectory can start from the original network and be terminated once the network can satisfy the demand under the reliability policy when the agent only adds capacity. Third, it can benefit from the stateful failure checking (§5) for a faster learning process. If a network survives a failure, then a network with more capacity can guarantee to survive the same failure. Thus, there is no need to check the failures that have been already survived. In contrast, we have to check all the failures on every step if reducing capacity is also included in the actions.

One domain-specific customization for action representation is that we use an action mask to handle the spectrum consumption constraints. The mask turns off the IP links if adding more capacity to these links would violate the spectrum consumption constraints. The stochastic policy only samples among valid IP links instead of all IP links.

Reward representation. The goal of NeuroPlan is to minimize the cost of the network while satisfying the traffic demand under the reliability policy. The ultimate reward is the cost of a network plan. While in principle we could return the cost of a network plan as a single final reward to the agent after generating a feasible solution, it would be hard to train the agent effectively, especially considering the cases with long trajectories where thousands of steps are needed for a feasible solution. To generate dense rewards, we assign an intermediate reward to each step with the cost of the newly added capacity and scale it down with a normalized parameter to get a final reward in the range of $[-1, 0)$. It is a common practice to use reward scaling to get better results for deep RL [21]. After a pre-defined number of steps, if we cannot get a feasible solution, we add -1 as the extra penalty for the final reward.

Training algorithm The RL agent is trained by an Actor-Critic algorithm [31]. In an Actor-Critic algorithm, we learn an actor $\pi(a|G; \theta, \theta_g)$ which gives a probability distribution of next-step action a given the current topology G, and a critic $V(G; \theta_v, \theta_g)$ that outputs a value to evaluate the current topology G. Actor-Critic algorithm is known to be more stable and efficient than the simple policy-gradient algorithm [27], and has been successfully applied to solve many tasks [3, 63, 69].

Algorithm 1 shows the pseudocode of the training algorithm. The algorithm first initializes the parameters of the actor, the critic, the GNN and the number of epochs (Line 2-3). For each epoch, several network plans are sampled from the probability distribution of the action returned by the current actor. With the same actor, many different plans can be sampled in order to achieve adequate exploration.

The generation of every network plan is termed as a trajectory. For each trajectory, NeuroPlan starts from the network topology with the original link capacity (which could be zero). It then generates the network plan by iteratively performing an action computed by the actor to the current network topology until the trajectory ends (Line 8-15). The trajectory is terminated under three conditions: (*i*) the current network topology satisfies the traffic demand under the reliability policy; (*ii*) the trajectory length exceeds a pre-defined threshold, and (*iii*) the trajectory is cut off by the current epoch. Any of the three conditions is true means the trajectory termination.

At the end of each epoch, we first compute the policy gradient loss (Line 18). The gradient loss of the actor is defined as the mean error between the advantage estimate and $log\ p$ across the epoch, where $log\ p$ is the logit of the corresponding sampled action. The advantage estimate for step i (GAE_i) is calculated as the GAE-Lambda advantage [53] by

$$GAE_i = r_i + \gamma \cdot v_{i+1} - v_i + \gamma \cdot \lambda \cdot GAE_{i+1}, \tag{6}$$

where r_i and v_i is the reward and the output of critic at step i respectively, γ is the discount factor and λ is a smoothing parameter for reducing variance. Then we compute the critic gradient loss (Line 21). It is defined as the mean-square error between the reward-to-go and v across the epoch, where v is the output of critic. The reward-to-go is calculated by applying the discount factor to the intermediate rewards.

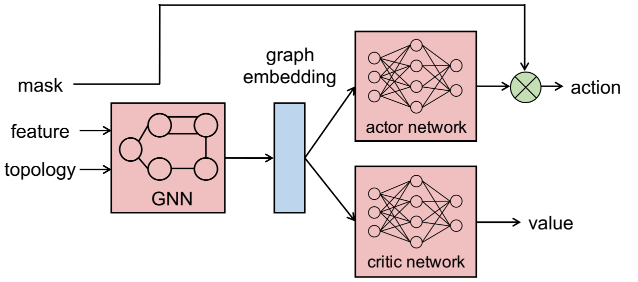


Figure 6: Actor and Critic network architecture of NeuroPlan.

Neural network architecture. Figure 6 shows the neural network architecture of the agent. We use a Graph Convolutional Network (GCN) [29] to encode the transformed network topology and generate a graph embedding. GCNs are a well-studied type of GNNs that achieve good performance in many graph representation tasks [29, 43, 74]. We have also experimented NeuroPlan with a Graph Attention Network (GAT) [64]. GATs introduce an attention mechanism as a substitute for the statically normalized convolution operation in GCNs. GATs did not perform as well as GCNs for our problem. Moreover, GAT has larger memory requirement, making it infeasible for large-scale problems.

The high-level idea of GCNs is to perform message propagation between neighbor nodes for multiple layers. Given n nodes and an f-dimension feature vector for each node, the first-layer graph embedding is represented as an $n \times f$ matrix, $H^{(0)}$. Then the $l+1$-th layer graph embedding can be computed by

$$H^{(l+1)} = ReLU(D^{-0.5}(A + I)D^{0.5}H^{(l)}W^{(l)}), \tag{7}$$

where $W^{(l)}$ is the learnable weight matrix at l-th layer, A is the adjacent matrix, I is the identity matrix, and D is the degree matrix of A. After L layers of GCN, we get the final graph embedding $G = H^{(L)}$. Then, we feed the graph embedding to the critic and actor network.

The critic and actor network are both simple Multilayer Perceptron (MLP). To realize a stochastic policy, the actor outputs the logits, which can be used to sample a feasible action after applying the action mask.

4.3 Search Space Pruning

As we have discussed in §4.1, considering the scale of the problem, it is challenging for deep RL to directly generate the final plan. Algorithm 1 indicates that the RL agent will keep adding capacity to the network until the network satisfies the traffic demand under the reliability policy. There may be useless steps in a feasible trajectory that do not contribute to satisfying the traffic demand. Thus, we use a two-stage hybrid approach which encodes the plan generated by deep RL as the maximum capacity constraints for the IP links to the ILP model. We then solve the ILP model to find the optimal solution under these constraints with an off-the-shelf ILP solver. To alleviate the impact of local optimum, we relax the maximum capacity constraints by multiplying the maximum capacities by the relax factor α. The relax factor α provides a tunable trade-off between optimality and tractability.

Interpretability of the solution. In essence, the two-stage approach resembles the existing approach described in §3.2. The RL agent takes over the job of the heuristics to prune the search space, and does so in an automated way without the need of human experts. Then ILP is applied to find the optimal solution in the pruned search space.

A common issue for deep learning approaches is interpretability. Our approach avoids this issue and makes the generated solution *interpretable*. Specifically, network operators can examine the solution from the RL agent and check whether the changes match their intuition and experience. More importantly, they can use the relax factor α to control the trade-off between optimality and tractability. This control is simpler and more explicit than tuning several hand-designed heuristics. It also clearly dictates the final solution is optimal in which part of the search space. And it is easy to incorporate additional modifications to the pruned search space from other heuristics.

5 IMPLEMENTATION

RL algorithm and environment. We implement the Actor-Critic algorithm of the RL agent based on the SpinningUp [55] framework and add support for GPU training. The RL environment is implemented in Python for compatibility. We list the hyperparameters used in NeuroPlan in Table 2.

Hyperparameter	Value
Max length per trajectory	{1024, 2048, 4096, 8192}
Max epochs to train	1024
Max length per epoch	{1024, 2048, 4096, 8192}
Max capacity units per step	{1, 4, 16}
Model nonlinearity	ReLU
GNN type	GCN
Number of GNN layers	0, 2, 4
MLP hidden layers	{64x64, 256x256, 512x512}
Actor learning rate	0.0003
Critic learning rate	0.001
Relax factor α	{1, 1.25, 1.5, 2}
Discount factor γ	0.99
GAE Lambda λ	0.97

Table 2: NeuroPlan hyperparameters.

Optimizations for the plan evaluator. As shown in Figure 3, the plan evaluator interacts with the RL agent and checks if the network plan satisfies the service expectations. For the problem of network planning, we only need to focus on the macro-scale behavior of the network (e.g., the IP link capacity). Thus, we do not need packet-level simulators (e.g., NS-3 [49]) that model micro-scale network behaviors such as network congestion. We do not need network emulators, either, such as Mininet [42] and CrystalNet [36] which run actual control plane and data plane code.

To check if the current network plan satisfies the service expectations, we formulate the problem as an LP problem and solve it with the Gurobi Optimizer [19]. Note that the LP problem is only to check if the current network plan can survive the failures and the network plan is given, which is much simpler than the existing approach (§3) for the entire network planning problem which tries to solve an ILP problem and find a plan with minimum cost. Gurobi Optimizer is a commercial optimization solver and supports a variety of programming languages. For efficiency, the failure checking part is implemented with Gurobi in C++ and compiled to a binary file which is then called in Python code. The implementation of NeuroPlan has two optimizations to accelerate the training process.

The first one is *source aggregation*. Assume there are l IP links, m IP nodes, n optical fibers and f flows. We have l variables to represent the IP link capacities and n constraints for spectrum consumption. For each failure, there are fm constraints for flow conservation, $2l$ constraints for IP link capacity (two directions for every IP link), $2l$ variables to represent the traffic volume on the IP links (two directions for every IP link). We do not need the spectrum consumption constraints when doing failure checking since they have been encoded in the action mask (§4.2). Given s failures, the total number of constraints is $s(fm + 2l)$, and the number of variables is $l + 2sfl$. If we do not consider the Classes of Services (CoS) for the flows, f can be m^2 at most. Rather than formulating the constraints with individual flows, we apply *source aggregation* [60], which aggregates the flows with the same source as one single flow. Such optimization decreases the number of constraints from $s(fm+2l)$ to $s(m^2+2l)$ and the number of variables from $l + 2sfl$ to $l + 2sml$.

The second one is *stateful failure checking*. As described in §4.1, if a network survives a failure, then a network with more capacity can guarantee to survive the same failure. To realize the stateful failure checking, we maintain a fixed order of failures. For each step, we check the failure scenarios starting at the failure that is not survived in the previous step instead of trying to solve all the failures at one time. Moreover, we can group the failures and employ multiple machines to check failure groups in parallel, which enables training for problems with a large number of failures. In some cases, the time to build up a Gurobi model is even longer than the time to solve the model. Thus, we only update the constraints that are influenced by the failure in the model, avoiding building up the model from scratch for each failure.

The evaluation results in §6.1 shows the efficiency of the two optimizations.

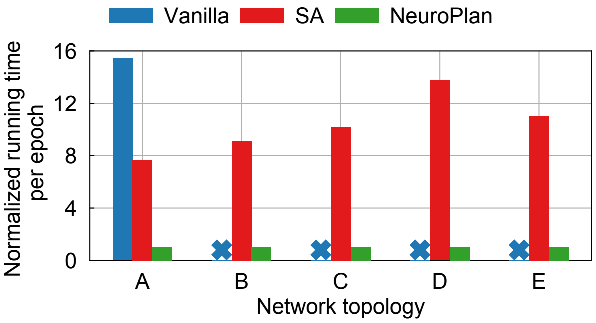


Figure 7: Implementation efficiency. The cross indicates the omitted entries in which the running time is larger than 2 hours. The running time is normalized with that of NeuroPlan on each topology.

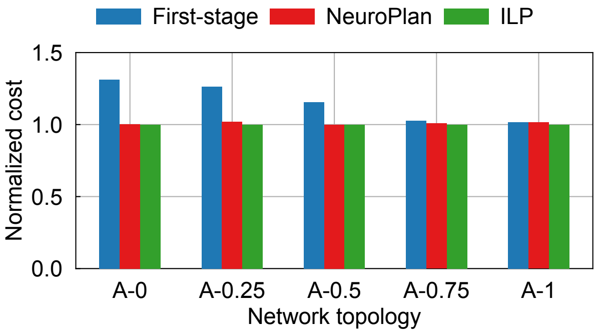


Figure 8: NeuroPlan can produce optimal solutions for small-scale problems. The cost results are normalized with that of *ILP* on each topology.

Workload patterns. The computation of neural networks, including forward propagation and backward propagation, takes significant time during training process. The backward propagation is done with the data of the entire epoch. To exploit the capability of AI accelerators (e.g., GPU) to accelerate training, we take a larger capacity increment unit so that RL agent can get feasible solutions with fewer steps and the data of the entire epoch can be fit into the GPU memory.

6 EVALUATION

In the evaluation, we aim to answer the following questions:

- How efficient is NeuroPlan with the optimizations on the implementation (§6.1)?
- How is the optimality of the network plan generated by NeuroPlan for small-scale problems (§6.2)?
- How is NeuroPlan compared with the heuristic approach for large-scale problems (§6.3)?
- What is the impact of the parameters (§6.4)?

We run NeuroPlan on g4dn.4xlarge AWS instances, with 16 CPU cores and one NVIDIA Tesla T4 GPU. We use five production network topologies with different scales, i.e., A, B, C, D and E, listed in the ascending order of topology size. In terms of the scale, A has tens of IP links, tens of failures and tens of (site-to-site) flows, and needs to add a few Tbps capacity for a feasible solution; E has hundreds of IP links, hundreds of failures and about one thousand flows, and needs to add a few hundred Tbps capacity for a feasible solution. We use real failure scenarios, traffic demands, reliability policies and cost models from production networks. We run NeuroPlan for each problem until convergence or for 1024 epochs.

We compare NeuroPlan with two alternative approaches: *ILP* and *ILP-heur*. *ILP* follows the problem formulation in §3.1 and solves it without any heuristics. *ILP-heur* integrates hand-designed heuristics into *ILP* to trade optimality for tractability. For faithful comparison, we use the same heuristic setups used in the production network. We also report the first-stage results generated by the RL agent (*First-stage*).

6.1 Implementation Efficiency

To show the benefits of the two implementation optimizations (i.e., source aggregation and stateful failure checking), we compare the running time of different implementations, including the vanilla plan evaluator (*Vanilla*), the plan evaluator with source aggregation (*SA*), and the plan evaluator with both source aggregation and stateful failure checking (NeuroPlan). We measure the average running time for 10 epochs on the five topologies (A, B, C, D, E). The running time is normalized with the running time of NeuroPlan on each topology. We omit the entries of *Vanilla* in which the running time is larger than 2 hours. As shown in Figure 7, *SA* reduces the running time by a factor of 2 with topology A. The efficiency of *SA* is further for other topologies (B, C, D, E) with more flows, as *SA* can reduce the number of constraints significantly. NeuroPlan is 7-14 times faster than *SA* as it applies the stateful failure checking.

6.2 Optimality for Small-Scale Problems

ILP can generate the optimal solutions for small-scale problems. We compare NeuroPlan and *ILP* on small-scale problems to evaluate the optimality of NeuroPlan. We vary the original capacities of the small topology A to create multiple synthetic problems with different sizes of search space. Specifically, A-0, A-0.25, A-0.5, A-0.75 and A-1 mean that the original capacity of each link is 0%, 25%, 50%, 75% and 100% of that of the corresponding link on topology A, respectively. We set the relax factor α to be 2. As shown in Figure 8, the results of *First-stage* is already close to the optimal with 75% (A-0.75) and 100% (A-1) original capacities. Even with 0% (A-0) original capacity where the RL agent generates the plan from scratch, the cost results of *First-stage* is no more than about 30% of the optimal cost. After the second stage, NeuroPlan is able to produce the final plan with a cost no more than 2% of the optimal cost.

6.3 Scalability for Large-Scale Problems

NeuroPlan can overcome the scalability issue encountered by *ILP*. Figure 9 compares the results of *First-stage*, NeuroPlan, *ILP-heur*

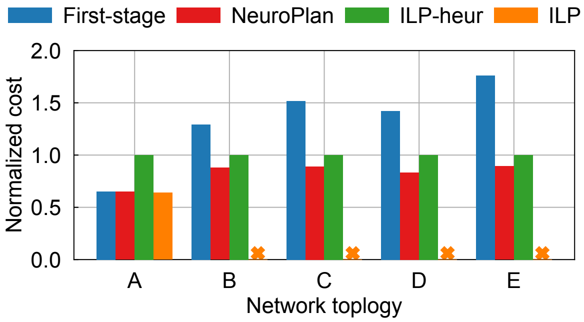


Figure 9: *ILP* fails to scale to large topologies, indicated by the crosses. NeuroPlan outperforms *ILP-heur* on large topologies and avoids human efforts to tune the heuristics. The cost results are normalized to that of *ILP-heur* on each topology.

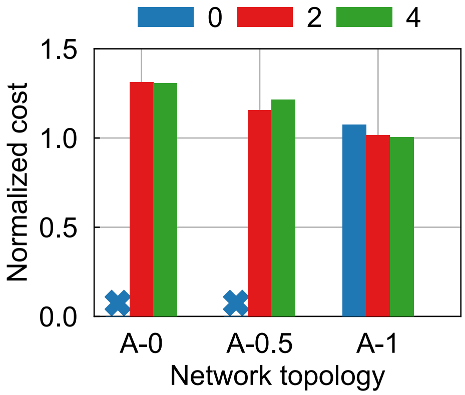
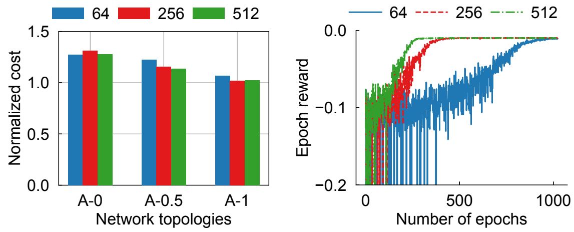


Figure 10: Impact of GNN layers on the results of *First-stage*. The crosses indicate that the RL agent does not converge. The results are normalized to the optimal cost on each topology.



(a) Impact of hidden size in MLP on cost results of *First-stage*.

(b) Impact of hidden size in MLP on convergence speed with A-1.

Figure 11: Impact of MLP on cost results of *First-stage* and convergence speed. The cost results are normalized with the optimal cost on each topology.

and *ILP* for the five topologies, i.e., A, B, C, D and E. The relax factor α is set to be 1.5. The results are normalized with the results of *ILP-heur*. *ILP* can only solve the problem with topology A, as the scale of other topologies is much larger than A and *ILP* cannot solve them in a reasonable amount of time (e.g., a few weeks). *ILP-heur* fails to achieve a good trade-off between optimality and tractability for all the topologies as different problems need customized heuristics for good results. For example, *ILP-heur* over-trades optimality for tractability on topology A as topology A is simple and can be solved completely with *ILP* for the optimal solution. In contrast, NeuroPlan prunes the search space automatically and specifically for each problem. After getting the pruned search space, NeuroPlan can solve it and get the optimal solution easily at the second stage. In addition, Figure 9 shows that NeuroPlan can get a plan for large network topologies (i.e., B, C, D and E) with a lower cost from 11% to 17% compared with that of *ILP-heur*.

6.4 Sensitivity Analysis

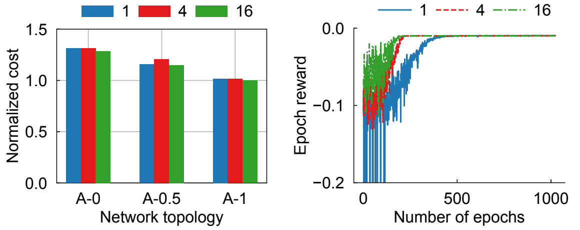
To better understand the impact of different parameters of NeuroPlan, we conduct a sensitivity analysis on three synthetic topologies used in §6.2, i.e., A-0, A-0.5, A-1. The parameters include the number of GNN layers, the hidden size of MLP, the maximum capacity unit per step and the relax factor α.

Figure 10 shows the costs of *First-stage* normalized with their corresponding optimal cost. Interestingly, NeuroPlan can learn even without GNN for A-1. However, it fails to learn and converge for other topologies (i.e., A-0 and A-0.5). It indicates that MLP alone can handle simple problems while GNN becomes essential for problems with a larger scale. Two or four layers of GNN have similar results. We also observed that the number of GNNs does not influence the convergence speed with respect to number of epochs (Figure 12(b)).

Then we vary the hidden size of MLP used by the critic and actor from 16x16 all the way up to 512x512. We report the cost results of *First-stage*. All the costs are normalized to the optimal cost on each topology. As shown in Figure 11(a), the MLPs with different hidden sizes converge to similar results. In Figure 11(b), we show the epoch reward v.s. the number of epochs for A-100. It indicates that larger hidden size leads to a faster convergence speed with respect to the number of epochs. This is because a more complicated MLP is able to model more complicated relationship and can be trained more efficiently. The results on A-0 and A-0.5 are similar.

We also study the impact of the maximum capacity unit per step. Figure 12(a) indicates that the maximum capacity unit per step has nearly no influence on the results of *First-stage*. Figure 12(b) shows the epoch reward v.s. the number of epochs for A-100. A larger maximum capacity unit leads to a faster convergence speed with respect to the number of epochs. We have also tried other topologies and found that this statement does not hold for other topologies. A larger maximum capacity unit only benefits the problems where the capacity increments are concentrated on a few links rather than scattered among many links in the converged solutions.

At last, we vary the relax factor α. To test a wide range of problem scales, we evaluate NeuroPlan with different α (i.e., 1, 1.25, 1.5) for the five topologies, i.e., A, B, C, D, E. Figure 13 reports the results of NeuroPlan normalized by the results of *First-stage*. The second stage does not improve the results a lot for topology A as RL already



(a) Impact of maximum capacity unit per step on results of *First-stage*.

(b) Impact of maximum capacity units per step on convergence speed with A-1.

Figure 12: Impact of maximum capacity units on results of *First-stage* and convergence speed. The cost results are normalized with the optimal cost on each topology.

gets a solution close to the optimal, while it does find better results up to 46% lower compared to the results of *First-stage* for other topologies. And it shows that with a larger α, NeuroPlan can get a better solution in a larger search space. Thus, the relax factor provides a convenient and tunable knob for the trade-off between optimality and tractability.

7 RELATED WORK

Network cost optimization. Minimizing the network cost while satisfying the service expectations is an important problem in network management. There is a lot of work on reducing the network cost by traffic engineering in both the IP layer [20, 23, 79] and the optical layer [37, 56]. Besides traffic engineering, capacity planning is also critical to the performance, reliability and cost of a network. A multi-layer capacity planning is proposed [15] to achieve minimum cost with a focus on multi-layer restoration. NeuroPlan focuses on the network planning problem and minimizes the network cost with GNNs and deep RL.

Graph neural network and RL. GNNs [33, 52, 70, 71, 77], as an efficient graph representation method, are widely used in many fields, including text classification [51], molecular feature extraction [12], protein structure prediction [1], chip design [43], and node classification [29]. Existing work also combines GNNs with RL for different tasks, such as solving classical combinatorial problems [28, 32, 45] and generating molecule graphs [74]. NeuroPlan exploits the capability of GNNs to encode the network topology and uses RL to efficiently explore the state space and prune the search space for ILP problems.

RL for optimization problems. Many solutions have applied RL to optimization problems [4, 5, 10, 40], including learning to cut for integer programming [58], learning branching policies for Mixed Integer Linear Programming [7, 18], building ML compilers [22, 61] and optimizing neural network architectures [25, 35, 80]. NeuroPlan makes two unique contributions to solving the network planning problem. First, it uses GNNs and a domain-specific node-link transformation for modeling dynamic network topologies. Second, it combines RL with ILP to address the scalability challenge for large-scale problems.

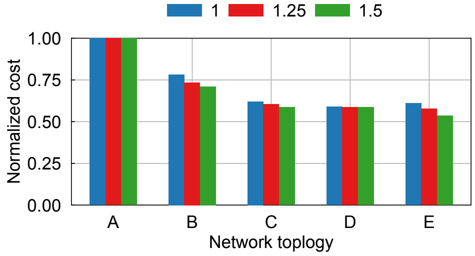


Figure 13: Impact of relax factor α. The cost results are normalized with those of *First-stage* on each topology.

Deep learning for networking and systems. Deep learning is an emerging tool to be used for solving networking and systems problems. NeuroCuts [34] uses Deep RL to generate packet classification tree with smaller depth and less memory footprint. AuTO [9] scales deep RL for datacenter-scale automatic traffic optimization. Pensieve [38] learns adaptive bitrate algorithms for better video quality. Metis [41] provides interpretability for deep learning-based networking systems. Valadarsky *et al.* [62] uses deep RL for better routing configurations. Yeo *et al.* [62] utilizes deep learning to reduce the dependency for delivering high-quality video. Deep learning is also used for congestion control [24], data layouts in big data systems [72], cache admission policy in content delivery networks [30], network resource management [57, 75, 76], scheduling algorithms [17, 39], concurrency control [67] and verification of distributed protocols [73]. NeuroPlan applies deep RL and combines it with GNNs to solve the network planning problem.

8 CONCLUSION

We present NeuroPlan, a deep RL approach to solve the network planning problem. NeuroPlan uses a GNN and a domain-specific node-link transformation for state encoding, and leverages a two-stage hybrid approach to find the optimal solution. The evaluation results show it can reduce the network planning cost by up to 17% compared with hand-tuned heuristics. Meanwhile, it avoids heavy human efforts to trade-off between optimality and tractability, and is easy to incorporate other heuristics.

Self-driving networks are an emerging trend. NeuroPlan makes a concrete step towards this goal by bringing AI techniques to an important network management task—network planning. We hope NeuroPlan can inspire more work of applying AI techniques to network automation.

This work does not raise any ethical issues.

Acknowledgments. We thank our shepherd Ratul Mahajan and the anonymous reviewers for their valuable feedback on this paper. Xin Jin (xinjinpku@pku.edu.cn) is the corresponding author. Xin Jin is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education. This work is supported in part by NSF grants CNS-1813487 and CCF-1918757, and Project 2020BD007 from PKU-Baidu Fund.

REFERENCES

[1] AlphaFold. https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology.

[2] Global state of the WAN Report, 2020. https://info.aryaka.com/state-of-the-wan-report-2020.html.

[3] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

[4] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[5] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[6] J. A. Bondy, U. S. R. Murty, et al. *Graph theory with applications*. Macmillan London, 1976.

[7] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *arXiv preprint arXiv:2006.01610*, 2020.

[8] Y. Chang, S. Rao, and M. Tawarmalani. Robust validation of network designs under uncertain demands and failures. In *USENIX NSDI*, 2017.

[9] L. Chen, J. Lingys, K. Chen, and F. Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *ACM SIGCOMM*, 2018.

[10] X. Chen and Y. Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 2019.

[11] CPLEX Optimizer. https://www.ibm.com/analytics/cplex-optimizer.

[12] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015.

[13] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[14] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *IEEE INFOCOM*, 2000.

[15] O. Gerstel, C. Filsfils, T. Telkamp, M. Gunkel, M. Horneffer, V. Lopez, and A. Mayoral. Multi-layer capacity planning for ip-optical networks. *IEEE Communications Magazine*, 2014.

[16] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[17] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin. Intelligent vnf orchestration and flow scheduling via model-assisted deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 2019.

[18] P. Gupta, M. Gasse, E. B. Khalil, M. P. Kumar, A. Lodi, and Y. Bengio. Hybrid models for learning to branch. *arXiv preprint arXiv:2006.15212*, 2020.

[19] Gurobi solver. https://www.gurobi.com/.

[20] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *ACM SIGCOMM*, 2015.

[21] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[22] Q. Huang, A. Haj-Ali, W. Moses, J. Xiang, I. Stoica, K. Asanovic, and J. Wawrzynek. Autophase: Juggling hls phase orderings in random forests with deep reinforcement learning. *arXiv preprint arXiv:2003.00671*, 2020.

[23] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM*, 2013.

[24] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, 2019.

[25] Z. Jia, M. Zaharia, and A. Aiken. Beyond data and model parallelism for deep neural networks. *arXiv preprint arXiv:1807.05358*, 2018.

[26] J. M. Kahn and K.-P. Ho. Spectral efficiency limits and modulation/detection techniques for dwdm systems. *IEEE Journal of Selected Topics in Quantum Electronics*, 2004.

[27] S. M. Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems*, 2001.

[28] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 2017.

[29] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[30] V. Kirilin, A. Sundarrajan, S. Gorinsky, and R. K. Sitaraman. Rl-cache: Learning-based cache admission for content delivery. *IEEE Journal on Selected Areas in Communications*, 2020.

[31] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 2000.

[32] W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[33] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[34] E. Liang, H. Zhu, X. Jin, and I. Stoica. Neural packet classification. In *ACM SIGCOMM*. 2019.

[35] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[36] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan. CrystalNet: Faithfully emulating large production networks. In *ACM SOSP*, 2017.

[37] Y. Liu, H. Zhang, W. Gongt, and D. Towsley. On the interaction between overlay routing and underlay routing. In *IEEE INFOCOM*, 2005.

[38] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM*, 2017.

[39] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh. Learning scheduling algorithms for data processing clusters. In *ACM SIGCOMM*, 2019.

[40] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *arXiv preprint arXiv:2003.03600*, 2020.

[41] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu. Interpreting deep learning-based networking systems. In *ACM SIGCOMM*, 2020.

[42] Mininet. http://mininet.org.

[43] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

[44] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 2002.

[45] A. Mittal, A. Dhawan, S. Manchanda, S. Medya, S. Ranu, and A. Singh. Learning heuristics over large graphs via deep reinforcement learning. *arXiv preprint arXiv:1903.03332*, 2019.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[47] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[48] T. Nishizeki and N. Chiba. *Planar graphs: Theory and algorithms*. Elsevier, 1988.

[49] NS-3 network simulator. https://www.nsnam.org/.

[50] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 1991.

[51] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *WWW*, 2018.

[52] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2008.

[53] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[54] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 2017.

[55] OpenAI Spinning Up. https://spinningup.openai.com/en/latest/.

[56] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, A. Cabellos-Aparicio, and P. Barlet-Ros. Routing in optical transport networks with deep reinforcement learning. *IEEE/OSA Journal of Optical Communications and Networking*, 2019.

[57] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos. Learning to optimize: Training deep neural networks for wireless resource management. In *IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2017.

[58] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, 2020.

[59] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and C. L. Zitnick. Elf opengo: An analysis and open reimplementation of alphazero. *arXiv preprint arXiv:1902.04522*, 2019.

[60] M. Tornatore, G. Maier, and A. Pattavina. Wdm network design by ilp models based on flow aggregation. *IEEE/ACM Transactions on Networking*, 2007.

[61] M. Trofin, Y. Qian, E. Brevdo, Z. Lin, K. Choromanski, and D. Li. Mlgo: a machine learning guided compiler optimizations framework. *arXiv preprint arXiv:2101.04808*, 2021.

[62] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar. Learning to route with deep rl. In *NIPS Deep Reinforcement Learning Symposium*, 2017.

[63] K. G. Vamvoudakis and F. L. Lewis. Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem. *Automatica*, 2010.

[64] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[65] S. Verdú. Spectral efficiency in the wideband regime. *IEEE Transactions on Information Theory*, 2002.

[66] S. Verdú and S. Shamai. Spectral efficiency of cdma with random spreading. *IEEE Transactions on Information Theory*, 1999.

[67] J. Wang, D. Ding, H. Wang, C. Christensen, Z. Wang, H. Chen, and J. Li. Polyjuice: High-performance transactions via learned concurrency control. In *USENIX OSDI*, 2021.

[68] P. J. Winzer. High-spectral-efficiency optical modulation formats. *Journal of Lightwave Technology*, 2012.

[69] Y. Wu and Y. Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *ICLR*, 2016.

[70] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[71] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[72] Z. Yang, B. Chandramouli, C. Wang, J. Gehrke, Y. Li, U. F. Minhas, P.-Å. Larson, D. Kossmann, and R. Acharya. Qd-tree: Learning data layouts for big data analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

[73] J. Yao, R. Tao, R. Gu, J. Nieh, S. Jana, and G. Ryan. Distai: Data-driven automated invariant learning for distributed protocols. In *USENIX OSDI*, 2021.

[74] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, 2018.

[75] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo. Resource management at the network edge: A deep reinforcement learning approach. *IEEE Network*, 2019.

[76] C. Zhang, P. Patras, and H. Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials*, 2019.

[77] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[78] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[79] D. Zhuo, M. Ghobadi, R. Mahajan, A. Phanishayee, X. K. Zou, H. Guan, A. Krishnamurthy, and T. Anderson. RAIL: A case for redundant arrays of inexpensive links in data center networks. In *USENIX NSDI*, 2017.

[80] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.