Enabling Robust DRL-Driven Networking Systems via Teacher-Student Learning

Ying Zheng, *Student Member, IEEE*, Lixiang Lin, Tianqi Zhang, Haoyu Chen, Qingyang Duan, Yuedong Xu, and Xin Wang

***Abstract*— The past few years have witnessed a surge of interest towards deep reinforcement learning (DRL) in computer networks. With extraordinary ability of feature extraction, DRL has the potential to re-engineer the fundamental resource allocation problems in networking without relying on pre-programmed models or assumptions about dynamic environments. However, such black-box systems suffer from poor robustness, showing high performance variance and poor tail performance. In this work, we propose a unified Teacher-Student learning framework that harnesses rich domain knowledge to improve robustness. The domain-specific algorithms, less performant but more trustable than DRL, play the role of teachers providing advice at critical states; the student neural network is steered to maximize the expected reward as usual and mimic the teacher's advice meanwhile. The Teacher-Student method comprises of three modules where the confidence check module locates wrong decisions and risky decisions, the reward shaping module designs a new updating function to stimulate the learning of student network, and the prioritized experience replay module to effectively utilize the advised actions. We further implement our Teacher-Student framework in existing video streaming (Pensieve), load balancing (DeepLB), and TCP congestion control (Aurora). Experimental results manifest that the proposed approach reduces the performance standard deviation of DeepLB by 37%; it improves the 90th, 95th, and 99th tail performance of Pensieve by 7.6%, 8.8%, and 10.7% respectively; and it accelerates the growth rate of Aurora by 2x at the initial stage, and achieves a more stable performance in dynamic environments.**

***Index Terms*— Deep reinforcement learning, networking systems, teacher-student learning.**

I. Introduction

INSPIRED by recent successes of deep reinforcement learning (RL) in the machine learning community [1], [2], the conventional computer networking field begins to embrace this new technique to conquer resource allocation problems. Representative applications include job scheduling [3], [4], congestion control [5], adaptive video streaming [6]–[9], and routing [10], [11]. The advantages of DRL in networking are twofold. One is the excellent feature extraction ability in dynamic environments. For instance, the job arrival rate and job size are stochastic, the bandwidth available to a user is time-varying. They cannot be accurately predicted by simple machine learning models because their dynamics is further influenced by various network functionalities. Another intriguing merit is that DRL offers an end-to-end mechanism for designing networking systems. Simply taking the observable state as input to neural networks, DRL is able to explore very large design space and yield a control policy via a black-box. It lowers down the technical know-how requirement of network operators, thus accelerating the deployment of system prototypes.

Despite the improved performance, these learning-based systems operate in a *black-box* mode which relies on opaque data-driven models. The lack of interpretability makes it difficult to completely trust that these deep models will perform well in real environments, and to debug a problematic decision [12]. For instance, the DRL agent in load balancing may place a short job to an overwhelmed server, which is *wrong*; it may leave one or more servers idle, expecting the future arrival of jobs to be short, which is *risk-seeking*. The wrong decisions will degrade the overall performance of DRL, and the risky decisions, though beneficial to the average performance, are inclined to cause large performance variance or degrade tail performance [13]. Therefore, the network operators tend to distrust a decision-making process when the wrong decisions and performance tails have high-stake consequences.

In this paper, we tackle the *robustness* problem of DRL in representative networking systems. The basic idea is to harness the rich domain knowledge to guide the decision-making of DRL. Though imperfect and less performant, the domain-specific rules can be utilized to identify the problematic actions of DRL and provide action advice accordingly. What we pursue is not an *ad hoc* approach for each individual networking system, but a *general* framework across them. The desirable properties include: i) high confidence in which domain knowledge can diagnose the problematic actions; ii) modularity where this framework is not reliant on specific DRL training as well as inference algorithms;

Manuscript received June 16, 2021; revised September 11, 2021; accepted October 31, 2021. Date of publication November 10, 2021; date of current version December 17, 2021. This work was supported in part by the Natural Science Foundation of China under Grant 61772139 and Grant 62072117 and in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010166003. *(Corresponding author: Yuedong Xu.)*

Ying Zheng, Haoyu Chen, and Xin Wang are with the School of Computer Science, Fudan University, Shanghai 200438, China (e-mail: zhengy18@fudan.edu.cn; haoyuchen17@fudan.edu.cn; xinw@fudan.edu.cn).

Lixiang Lin, Qingyang Duan, and Yuedong Xu are with the School of Information Science and Technology, Fudan University, Shanghai 200438, China (e-mail: lxlin19@fudan.edu.cn; ydxu@fudan.edu.cn).

Tianqi Zhang is with the School of Mathematical Science, Fudan University, Shanghai 200438, China (e-mail: tianqizhang18@fudan.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2021.3126085.

Digital Object Identifier 10.1109/JSAC.2021.3126085

iii) minimal intervention that the ratio of the actions advised by domain knowledge to the total actions should be kept small.

We model the DRL-empowered networking systems based on Markov decision processes (MDPs) with *input-driven environments* similar to [14]. The root cause of undesirable performance lies in the dynamic networking environment such as time-varying bandwidth and stochastic job arrivals. We define two types of actions, namely *wrong decision* and *risky decision* that affect the average performance and its variance in the input-driven MDPs. We theoretically show that the wrong decisions degrade average performance and the risky decisions increase performance variance. In practical networking systems, reinforcement learning is embraced to tackle the complicated Markov decision problems in a model-free mode. This imposes great difficulties to pinpoint the wrong and risky decisions precisely. Inspired by [15], we propose a novel and universal *Teacher-Student framework* to improve the robustness of DRL-based networking systems. The teacher is a small set of simple *white-box* logic rules that are specified by domain-specific algorithms or human engineers. The student is an independent DRL agent but is infused with teacher's advice. Three key components are developed, including confidence check for locating critical states, reward shaping for injecting teaching data into student network and prioritized experience replay for coping with the challenge of limited teaching data.

We implement our Teacher-Student framework in three systems: Pensieve [6] for video streaming, DeepLB for load balancing, and Aurora [5] for TCP congestion control. The teachers are chosen to be the buffer-based algorithm [16], the shortest processing time algorithm and TCP BBR [17], respectively. Comprehensive experiments manifest that our method improves the tail performance (almost) or reduces the variance without sacrificing the average performance, thus making the DRL agents more robust. In particular, the real-world evaluation in Pensieve shows that the 90th, 95th and 99th percentile QoE values are improved by 7.6%, 8.8% and 10.7%. When taking the shortest processing time algorithm as the benchmark, our method reduces the standard deviation of DeepLB job processing time by 37%. In NS3 simulations, our method improves the response speed of TCP Aurora (by nearly 2x faster) at the initial climb-up stage, and demonstrates gentle improvements in both the bandwidth utilization and the round-trip time in dynamic network environments.

The contributions are summarized below:

- We introduce the robustness problem of DRL-driven networking systems that is essentially originated from the lack of interpretability of deep neural networks and the high-dimensional state-action space of RL.
- We define the wrong and risky decisions for a RL agent, and theoretically show that the wrong decisions degrade average system performance, while risky decisions increase performance variance.
- A Teacher-Student framework is proposed to integrate domain knowledge to deep neural network based networking systems, in which the key components are (i) Confidence Check for locating critical states, (ii) Reward Shaping for incorporating domain knowledge, and (iii) Prioritized Experience Replay for efficient training.
- We evaluate the proposed framework in three classical systems: video streaming with both simulated and real-world implementations across four datasets, load balancing with a simulated system and TCP congestion control with an emulated system. The experimental results manifest the effectiveness of the proposed methods.

The rest of this paper is organized as follows. We first introduce the background in section II and motivation in section III-A. We present the system model in section IV, in which the wrong and risky decisions are modeled formally. In section V, we introduce three key components in the proposed Teacher-Student framework. We evaluate our algorithms and analyze the evaluation results in section VI. The related work is provided in section III-B. Some open problems about the proposed method are discussed in section VII. Finally, we conclude the paper in section VIII.

II. Preliminaries

A. Deep Reinforcement Learning

The standard RL problem is formalized as a discrete time MDP described by a tuple $< \mathcal{S}, \mathcal{A}, p, r, \gamma >$ [18], where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p(s'|s,a)$ represents the probability of entering state s' when the agent takes action a at state s. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. $\gamma \in [0,1]$ is a discounted factor that indicates how much the agent value an immediate reward compared to future rewards. $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$ is a policy that describes the probability distribution over all actions at a given state. $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \cdots)$ is a trajectory, and $\tau \sim \pi$ indicates the distribution over trajectories depends on π. In following sections, for the convenience of proof, we omit reward r in the trajectory τ and use the form of state-action pairs, i.e. $\tau = \{(s_0, a_0), (s_1, a_1), \cdots\}$.

The agent and the environment interact at discrete time steps, $t = 0, 1, 2, \cdots$. At each time step t, the RL agent observes state $s_t \in \mathcal{S}$, and selects an action $a_t \sim \pi(\cdot|s_t) \in \mathcal{A}$. Following this action, the RL agent receives a reward r_t and the state of environment transits to $s_{t+1} \sim p(\cdot|s_t, a_t)$. The goal of RL is to find the optimal policy π^* that maximizes the expected cumulative reward:

$$\max_{\pi} \quad \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] \tag{1}$$

The expectation is taken over a random policy π. The policy π defines a probability distribution based on which an action is chosen. On the contrary, a deterministic policy selects a single action per state. In DRL settings we often use a series of parameters, i.e. neural networks, to approximate policy π so that it is often rewritten as π_θ.

To solve the MDP problem, the concepts of state-value function and action-value function are introduced. The *state-value function* of a state s_t is the expected cumulative reward starting from s_t and thereafter following policy π. Formally,

$$v_\pi(s_t) = \mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})|s_t] \tag{2}$$

Similarly, the *action-value function* for policy π, $q_\pi(s_t, a_t)$, is defined as:

$$q_\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi}[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})|s_t, a_t] \quad (3)$$

The reinforcement learning algorithms can be divided into two categories, value-based and policy-based algorithms. The policy-based algorithms directly output the probability distribution based on which the action is chosen. The policy-based algorithms are suitable for both discrete and continuous actions, e.g. REINFORCE [19] and PPO [20]. The value-based methods output the value function of the state action pair, and the action corresponding to the highest value function is more likely to be selected. Value-based methods are only suitable for discrete actions such as Q-learning, DQN [2] and many variants [21]. Besides, the Actor-Critic algorithm combines policy-based and value-based methods. In this paper, we choose to study the PPO and Actor-Critic algorithms because of their good performance and wide adoption in networking.

B. DRL in Video Streaming

Adaptive streaming is the predominant form of video delivery service nowadays. A video is subdivided into a sequence of chunks, each of which contains 2∼4 seconds of content and is encoded in multiple bitrates. The higher bitrate means the better video quality. Client-side video players employ bitrate adaptation algorithms to automatically select the chunk bitrates that match the achievable throughput. For instance, the buffer-based algorithm (BBA), the classical rule based policy, uses the buffer length to guide the bitrate selection. When the client-side buffer length is below (above) the minimum (maximum) threshold, the lowest (highest) bitrate is requested. Otherwise, the requested bitrate increases roughly linearly to the buffer occupancy. However, simple logic rules such as BBA need delicate parameter configurations, like the two thresholds mentioned above. They are usually myopic to the current system state, while largely overlooking the dynamics of user throughput in the recent past and the deep interplay between the system state and the bitrate adaptation.

Pensieve [6] tackles this challenge by using DRL technique to generate bitrate selection. As shown in Fig. 1, Pensieve's state consists of recent chunk throughput, buffer sizes and actions. Pensieve's action is the bitrate of next video chunk. The goal of Pensieve is to maximize the quality of experience (QoE), a metric consisting of video quality and playback fluency that directly reflects the perception of users. Pensieve trains its neural network using the Asynchronous Advantage Actor Critic (A3C [22]) algorithm. See [6] for more details.

C. DRL in Load Balancing

In a load balancing system, there is one master and multiple working servers. The master appropriately allocates the incoming jobs according to a scheduling policy. Then the server would execute the allocated jobs in a first-in-first-out (FIFO) manner. An effective scheduling algorithm adjusts the scheduling decisions according to the running status of current

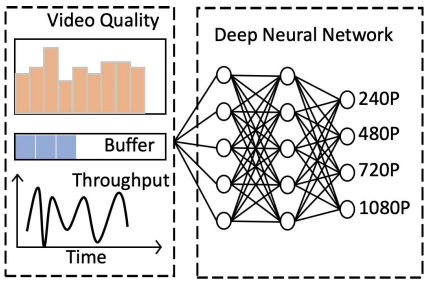


Fig. 1. An example state of Pensieve.

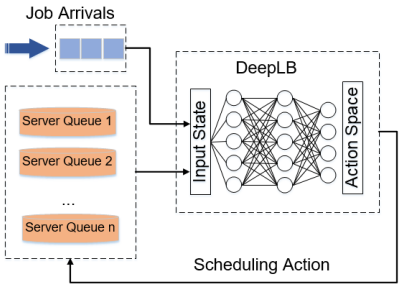


Fig. 2. An example state of DeepLB.

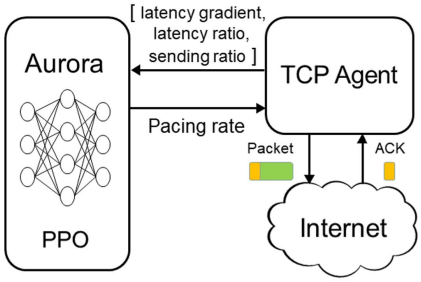


Fig. 3. An example state of Aurora.

system with the hope of minimizing average job completion time (JCT) consisting of waiting time and processing time. The conventional scheduling algorithms require experts to manually configure many parameters. Furthermore, the heuristic logic needs to be re-adjusted upon the changes of environments, e.g. the job arrival patterns. Currently a promising trend is to use deep neural network based algorithms to achieve end-to-end load balancing.

DeepLB: DeepLB is a DRL-based application that we create based on the load balancer in [14]. The difference between DeepLB and the original version is that the latter does not consider the elapsed time of processing the head of line job in each queue. Omitting this feature will cause the performance degradation in both the DRL model and in the heuristic algorithm. Therefore, we redesign the input state and train a new RL agent named DeepLB. Suppose there are k servers in the load balancing system. At each decision point t, the RL agent observes state $s_t = (j, q_1, q_2, \cdots, q_k)$, where j is the size of the incoming job, q_k is the sum of k^{th} queue size and the remaining size of the current executing job. The

RL agent choose an action $a \in \{1, 2, \cdots, k\}$, indicating the incoming job is allocated to a server, based on this observed state. This policy is often approximated by a neural network due to the huge state space. When a new job arrives at t_i, the reward is calculated by $r_i = -(t_i - t_{i-1}) \times n$, where n represents the number of active jobs in the system during period $[t_{i-1}, t_i]$, t_{i-1} is the timing of the last event. Both the arrival and completion of jobs will trigger the calculation of reward. In addition, DeepLB uses Actor-Critic algorithm to train its policy.

D. DRL in TCP Congestion Control

TCP congestion control is a natural test field of DRL. Owing to the complicated interaction between the senders and the dynamic environment, the traditional algorithms such as TCP CUBIC [23] and TCP BBR [17] may cause the bufferbloat or underutilize the bandwidth. DRL enables a TCP sender to learn its transmission rate from experience in which the up-to-date DRL congestion controller is TCP Aurora [5].

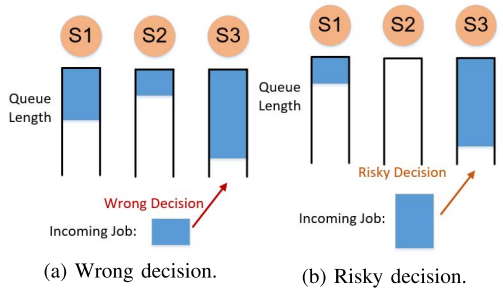
TCP Aurora uses Proximal Policy Optimization (PPO [20]) as the training algorithm. The state space contains a sequence of network states in the past k monitor intervals. Each network state is a 3-tuple: (i) latency gradient, the derivative of latency with respect to time; (ii) latency ratio, the ratio of mean latency at the current monitor interval to minimum observed mean latency in the history; and (iii) sending ratio, the ratio of transmitted packets to the acknowledged packets. The action space of Aurora is continuous, other than a finite enumerable action space to adjust congestion window size. The action is absorbed as a scaling factor to tune the source sending rate that is implemented through TCP pacing. The reward of a TCP sender is the linear combination of throughput, round-trip delay and packet loss ratio. In terms of their inner workings, BBR and Aurora are very similar, i.e. adjusting the sending rate according to the observed network status. The difference lies in that the former uses the bandwidth delay product to tune the sending interval at the packet granularity, and the latter uses a simple neural network to infer the sending rate at each time interval. Let x_t be the sending rate, a_t be the action at time t, and α be a constant. The control logic of Aurora is

$$x_t = \begin{cases} x_{t-1}(1 + \alpha a_t), & a_t > 0 \\ x_{t-1}/(1 - \alpha a_t), & a_t < 0. \end{cases} \tag{4}$$

III. Motivation

A. Motivating Examples

DRL agents make decisions via a black-box manner. The statistical performance gains often camouflage the deep understanding of their inner-workings. Kazak and Kazak in [24] verified multiple DRL-driven networking systems, exposing scenarios where undesirable behavior may appear. In addition, various counter-examples were generated accordingly. We hereby argue that the DRL models can be *wrong* and may take *risky* actions to gain more rewards. The wrong and risky decisions exist in almost all the networking applications in which the load balancer is a superb example for demonstration.



(a) Wrong decision. (b) Risky decision.

Fig. 4. Wrong decision and risky decision examples in load balancing.

1) Wrong Decision: The incoming short job is assigned to the worker where some long jobs have been waiting in the queue (Fig.4a). The length of the blue blocks represents the processing time of jobs. Since DeepLB is purposed to minimize the average processing time of all the jobs, a proper strategy is to place this short job to the server with a few short jobs, hence the previous action is a wrong decision.

2) Risky Decision: Suppose that one server is idle and the others have buffered jobs. An incoming long job is not assigned to this idle worker, but is placed at a queue already with long jobs in the front (Fig.4b). DeepLB makes this non-work-conserving decision by expecting the future arrivals would be short jobs.

The wrong decision and the risky decision play different roles in the decision making of DRL. The former lowers down the average performance, while the latter "entices" the agent to improve the average performance while possibly sacrificing the tails or increasing the variation of performance. Knowing these abnormal decisions requires profound *domain knowledge* that refers to either the direct human perception or the classical rule-based algorithms.

The domain knowledge in networking is far *imperfect*. It might tell a tiny fraction of the wrong decisions using only low-dimensional information (e.g. queue lengths in DeepLB and playback buffer length in Pensieve), and can hardly gauge risky decisions in more complicated scenarios. Meanwhile, the reward of the actions suggested by rule-based algorithms cannot be acquired *immediately*, e.g. the average JCT of DeepLB is measured for a sequence of jobs. In what follows, we aim to put forward a unified DRL framework that can harness the imperfect domain knowledge in networking systems.

B. Related Work

Research on the robustness of DRL-driven networking systems can be roughly divided into the following three categories:

- *Guided Exploration:* This type of work focuses on *training* a DRL agent with the participation of traditional algorithms. For example, Salma *et al.* proposed Eagle, which used a rule-based algorithm, BBR, to generate action advice and put it into the training of the DRL agent, thereby accelerating the training process and further optimize the decision policy [25]. Although Eagle achieves better performance than most of the comparing

algorithms, it has difficulty outperforming BBR algorithm that provides action advice. This is essentially different from our work, which aims to guide a more robust DRL agent even with the teaching algorithm inferior to the DRL student.

- *Interpretability Analysis:* Zheng *et al.* analyzed causality between input and output of a DRL-driven job scheduling system by inspecting the active status of inner neurons [12]. Meng *et al.* converted the learning-based networking systems into decision trees with clearer decision logic, reducing the long delay caused by the complex calculations of neural networks [26], [27].
- *Switching Between DRL Agent and Traditional Algorithms:* Soheil *et al.* proposed a DRL agent making decisions to adjust the congestion window at regular time intervals, and the Cubic algorithm was used in each time interval [28]. Mao *et al.* proposed to use a rule-based algorithm to replace the decisions made by DRL agent when changes over network conditions are detected, hence helping the DRL agent adapt to dynamically changing network environment faster [29]. However, the goal of our work is to *automatically incorporate domain knowledge* and hence enabling a more robust DRL agent. Thus, a new Teacher-Student framework is needed.

Apart from the aspect of networking systems, robust reinforcement learning has been comprehensively studied in the literature. The core of robust RL is to avoid or modify the actions taken by an agent at certain states so that it will not enter unsafe states [30]. Safe RL is usually realized via two approaches. One is modifying the risk-neutral objective function to be risk-sensitive. Instead of maximizing the expected return, [31], [32] adopt an expected-exponential criterion whose Taylor expansion is essentially a sum of expected return and variance of return. The variance minimization problem of a discrete-time discounted MDP is studied in [33], whose discounted performance is equal to a given constant. The risk of a policy can be transformed into one or more constraints to the original MDP problem so that the robust decision-making becomes a constrained MDP problem [34], [35]. Despite their theoretical soundness, these methods usually demand accurate MDP models that are hard to obtain in practical networking systems.

The other is providing external knowledge to adjust the exploration behavior of the agent. The external knowledge gathered from experts or previous demonstrations can be leveraged to generate a good "initialization" of learning algorithms, e.g. [36], [37]. The irrelevant regions of the states and action spaces will be explored less from the earliest steps of the learning process, thus accelerating the convergence of the learning algorithms. The external knowledge in the form of teacher advice can further be used to improve the exploration during the learning process. In such robust RL systems, there exist a teacher and a learning agent. At every step, the learning agent performs as usual, and the teacher monitors the learning process and interacts with the learning agent by providing action advice or related information. Clouse proposed an integration method of apprentice learning and reinforcement learning in which the learner can improve its policy based on rewards and on actions provided by human or a controller [38]. Garcia and Fernandez considered the RL problem with a dangerous and high-dimensional state-action space in which avoiding or minimizing damage caused by the explorations are emphasized [39]. Two new components for safe exploration are introduced: a risk function to evaluate the danger of a particular state and a baseline behavior that produces the suggested safe actions.

The Teacher-Student framework in this paper falls in the scope of external knowledge intervention. The teacher is not human, but a domain-specific algorithm that is able to identify a fraction of wrong or risky actions and provides suboptimal albeit safe actions.

IV. System Model

In this section, we first introduce the concept of MDP with an input stochastic process, after which the wrong and risky decisions are formally defined. Then we propose the theoretical analysis on the system performance influenced by these problematic decisions. Finally we present the overall Teacher-Student framework.

A. MDP With an Input Stochastic Process

For most of the networking systems driven by DRL, the state transition depends on not only state-action pairs, but also the dynamic of environment, e.g. network bandwidth, job arrivals, which is termed *input-driven environments*, resulting an input-driven MDP [14]. We use a discrete-time input process $\boldsymbol{z} = \{z(t), t = 0, 1, 2 \cdots\}$ to denote the external input process, which is unknown in prior. A sample path of the input process is denoted by $\boldsymbol{\omega} = (\omega_0, \omega_1, \cdots),\ \omega \in \Omega$, where Ω represents all possible sample paths. A graphic model of input-driven MDP is shown in Fig. 5. The choice of actions is the same as the standard MDP, which depends on policy $\pi(\cdot|s)$ and the current system state. While the calculation of state transition function $p(\cdot|s, a, z)$ and reward function $r(s, a, s', z)$ also need to consider the impact of the input value, in addition to the state-action pair.

With the influence of the input process $\boldsymbol{z} = \{z(t), t = 0, 1, 2 \cdots\}$, we rewrite the expression of state-value function $v_\pi(s_t)$ (Eq. (2)) and action-value function $q_\pi(s_t, a_t)$ (Eq. (3)) as

$$v_{\pi,\boldsymbol{z}}(s_t) = \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})|s_t], \tag{5}$$

where the expectation is taken over the random policy. Similarly, the action-value function for policy π, is defined as:

$$q_{\pi,\boldsymbol{z}}(s_t, a_t) = \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})|s_t, a_t]. \tag{6}$$

B. Wrong Decisions and Risky Decisions

In section III-A, we describe the motivating examples of wrong and risky decisions. We next present the formal definitions of them.

Definition 1 (Wrong Decisions): Let s represent the state observed by the agent at time step t, based on which the agent selects an action a. We call action a wrong decision at state s if there exists an another action a' such that $q_{\pi,\boldsymbol{z}}(s,a) < q_{\pi,\boldsymbol{z}}(s,a')$ holds for arbitrary π and $\boldsymbol{z}$. Furthermore, the corresponding (s,a) is called a wrong state-action pair.

Remark: From Definition 1 we obtain $\mathbb{E}_{\pi,\boldsymbol{z}}[q_{\pi,\boldsymbol{z}}(s,a)] < \mathbb{E}_{\pi,\boldsymbol{z}}[q_{\pi,\boldsymbol{z}}(s,a')]$. In other words, the wrong decision is not the optimal one when the agent regards maximizing the expected return as a goal.

Theorem 1: For a trajectory $\tau = ((s_0,a_0),(s_1,a_1),\cdots)$ with a set of wrong state-action pairs $D_\tau \subset \tau$, there always exists a set of state-action pairs D_c such that the agent can gain more cumulative reward by executing the state-action pairs in D_c instead of D_τ.

Proof: For the trajectory τ containing wrong state-action pairs, we divide it into two segments: $\tau_1 = ((s_0,a_0),(s_1,a_1),\cdots,(s_{m-1},a_{m-1}))$ and $\tau_2 = ((s_m,a_m),(s_{m+1},a_{m+1}),\cdots)$ with (s_m,a_m) is the first wrong state-action pair that appears in trajectory τ. Given an input value sequence $\boldsymbol{z}$, the expected cumulative reward can be calculated as

$$\begin{aligned} J_\pi &= \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)] \\ &= \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{t=0}^{m-1}\gamma^t r(s_t,a_t) + \sum_{t=m}^{\infty}\gamma^t r(s_t,a_t)] \\ &= \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{t=0}^{m-1}\gamma^t r(s_t,a_t)] + \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{t=m}^{\infty}\gamma^t r(s_t,a_t)] \\ &= \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{t=0}^{m-1}\gamma^t r(s_t,a_t)] + q_{\pi,\boldsymbol{z}}(s_m,a_m). \end{aligned} \tag{7}$$

Definition 1 indicates $\exists a'_m$ such that $q_{\pi,z}(s_m,a_m) < q_{\pi,z}(s_m,a'_m)$. Let $D_c = \{(s_m,a'_m)\}$, Eq. (7) becomes:

$$\begin{aligned} J_\pi &= \mathbb{E}_{\pi,\boldsymbol{z}}[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)] \\ &\le \mathbb{E}_{\pi,\boldsymbol{z}}\sum_{t=0}^{m-1}\gamma^t r(s_t,a_t) + q_{\pi,\boldsymbol{z}}(s_m,a'_m) \\ &= J_{\pi'}. \end{aligned} \tag{8}$$

Furthermore, we could separate τ_2 in the similar way, obtaining a D_c with more state-action pairs. Eq. (8) manifests that choosing state-action pairs in D_c would receive more expected cumulative reward. Theorem 1 indicates wrong decisions degrade overall average performance. □

Definition 2: Let $v_\pi(s)$ be the total reward in a realization of MDP starting from state s thereafter following policy π. The variance of an MDP with an input process $\{z(t), t = 0,1,2\cdots\}$ is defined as $\mathrm{Var}[v_{\pi,\boldsymbol{z}}(s)] = \mathbb{E}[v_{\pi,\boldsymbol{z}}(s)^2]$- $\mathbb{E}[v_{\pi,z}(s)]^2$.

Definition 3 (Risky Decisions): Considering an MDP with a discrete-time input process $\{z(t), t = 0,1,2\cdots\}$, the observed state at time step t is s, we call a' at state s is a risky decision with respect to a if $\mathbb{E}_{\boldsymbol{z}}[q_{\pi,\boldsymbol{z}}(s,a)] \le \mathbb{E}_{\boldsymbol{z}}[q_{\pi,\boldsymbol{z}}(s,a')]$, $\mathrm{Var}[r_{s,s'}|s,a] \le \mathrm{Var}[r_{s,s'}|s,a']$ for arbitrary $s' \in \mathcal{S}$, where $a, a' \in A_s$, and A_s is the available actions at state s.

Remark: According to Definition 3, there exists a special case that the action is a risky decision with respect to itself, i.e. $\mathbb{E}_{\pi,z}[q_{\pi,z}(s,a)] = \mathbb{E}_z[q_{\pi,z}(s,a')]$ and $\mathrm{Var}[r_{s,s'}|s_t,a] = \mathrm{Var}[r_{s,s'}|s,a']$ with $a = a'$. It is not our focus and we ignore this case. In addition, it can be inferred from Definition 3 that the risky decision is possible but not necessarily the optimal decision.

Proposition 1: Given two policy π and π', the only difference is that policy π' selects a risky decision a' while policy π selects the safer counterpart, i.e. action a, at state s deterministically, then we get $J_\pi \le J_{\pi'}$, i.e. $\mathbb{E}_{\pi,\boldsymbol{z}}\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) < \mathbb{E}_{\pi',\boldsymbol{z}}\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)$

We omit the proof of Proposition 1 here. It is natural that, from the definition of risky decisions, choosing a risky decision at a specific state s would gain more expected cumulative reward, thus better average system performance. Next we provide a formal definition of the variance of MDP with input processes.

Theorem 2: Given the current system state s, available action set A_s and input process $\{z(t), t = 0,1,2\cdots\}$, if $\exists a, a' \in A_s$, where action a' is a risky decision w.r.t. action a, choosing risky action a' increases the variance of MDP.

Proof: Let $\sigma^2_{\pi,\boldsymbol{z}}(s)$ represent the variance of cumulative reward starting from state s under the guidance of policy π and $\sigma^2_{\pi,\boldsymbol{z}}(s,s')$ represents the variance of reward received from the transition from state s to state s'. We rewrite $\sigma^2_{\pi,\boldsymbol{z}}(s)$ and $\sigma^2_{\pi,\boldsymbol{z}}(s,s')$ as σ^2_s, $\sigma^2_{ss'}$ for simplicity, respectively. In addition, we assume a finite state space $\mathcal{S}$ with $|\mathcal{S}| = N$ in the following proof.

From Eq. (6) in [40], we know that for an arbitrary state $s \in \mathcal{S}$, the following equation holds:

$$\sigma^2_s = \sum_{s' \in \mathcal{S}} p_{ss'}[\sigma^2_{ss'} + \gamma^2\sigma^2_{s'} + (r_{ss'} + \gamma v^2_{s'})] - v^2_s \tag{9}$$

This system of linear equations can be rewritten as the form $A\boldsymbol{x} = \boldsymbol{b}$, where the coefficient matrix $A = I - \gamma^2 P$, and P is the transition probability matrix. Specifically,

$$\begin{bmatrix} 1-\gamma^2 p_{11} & -\gamma^2 p_{12} & \cdots & -\gamma^2 p_{1N} \\ -\gamma^2 p_{21} & 1-\gamma^2 p_{22} & \cdots & -\gamma^2 p_{2N} \\ \vdots & & \ddots & \\ -\gamma^2 p_{i1} & -\gamma^2 p_{i2} & \cdots & -\gamma^2 p_{iN} \\ \vdots & & \ddots & \\ -\gamma^2 p_{N1} & -\gamma^2 p_{N2} & \cdots & 1-\gamma^2 p_{NN} \end{bmatrix} \begin{bmatrix} \sigma^2_1 \\ \sigma^2_2 \\ \vdots \\ \sigma^2_i \\ \vdots \\ \sigma^2_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_N \end{bmatrix} \tag{10}$$

where $b_i = \sum_{j=1}^{N} p_{ij}[\sigma^2_{ij} + (r_{ij} + \gamma v_j - v^2_i)]$. Since the transition probability matrix P is a row stochastic matrix, the largest eigenvalue of P is 1. Hence the coefficient matrix $A = I - \gamma^2 P$ does not have eigenvalue 0 because $\gamma \in (0,1)$ in an infinite horizon MDP. Thus the inverse of square matrix A exists, and $\boldsymbol{\sigma} = A^{-1}\boldsymbol{b}$ is the variance of states.

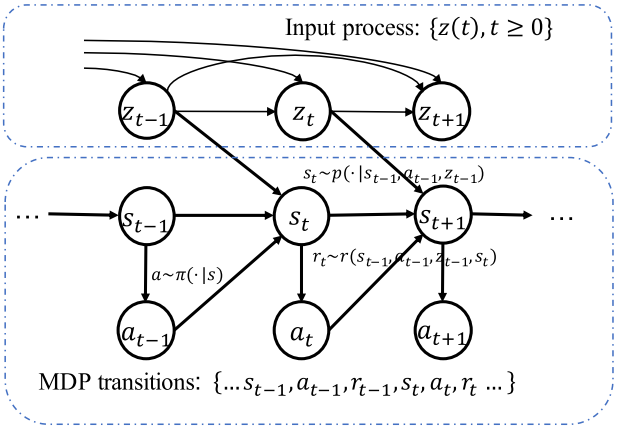


Fig. 5. Graphical model of an input-driven MDP.

Definition 3 manifests the risky decision has a larger variance compared with its safer counterpart. Formally, for a risky decision a' w.r.t. its safer counterpart a at state s, The variance of a risky decision $\sigma_{ij}^{2\,\prime} > \sigma_{ij}^2$, which incurs a positive increase of $\boldsymbol{b}$. When choosing risky action a', the variance of all states is calculated by:

$$\boldsymbol{\sigma}' = A^{-1}(\boldsymbol{b} + \Delta\boldsymbol{b}) = A^{-1}\boldsymbol{b} + A^{-1}\Delta\boldsymbol{b} = \boldsymbol{\sigma} + A^{-1}\Delta\boldsymbol{b}. \quad (11)$$

We next prove that A^{-1} is a non-negative matrix, i.e. all elements are non-negative values. The coefficient matrix A of the system of variance equations is a strictly diagonally dominant matrix, i.e. $|A_{ii}| > \sum_{i \neq j} |A_{ij}|$. Specifically, for an arbitrary row i in the coefficient matrix A,

$$\begin{aligned} \sum_{i \neq j} |A_{ij}| &= |-\gamma^2 \sum_{i \neq j} p_{ij}| \\ &= |-\gamma^2(1 - p_{ii})| \\ &< |1 - \gamma^2 p_{ii}| = |A_{ii}|. \end{aligned} \quad (12)$$

The last inequality is because $\gamma \in (0, 1)$. For the coefficient matrix A, $\forall i \neq j, A_{ij} < 0$, hence A is a Z-matrix. Furthermore, A is a M-matrix since the real parts of eigenvalues of the strictly diagonally dominant matrix A are positive. From [41] we know that, the inverse matrix of M-matrix has all non-negative elements, so we get $\forall i \in [N], \sigma_i^2 \leq \sigma_i'^2$, i.e. choosing risky decision a' increases the variance of MDP. □

C. Teacher-Student Model

The domain knowledge in networking applications, though imperfect, entails us to find wrong or risky decisions. A more robust learning system arises if these domain knowledge can be integrated into the learning systems. Inspired by [15], we adopt a novel Teacher-Student learning framework. The main idea is that RL agents maximize the expected cumulative return, as standard RL does, while minimizing the distance to its teaching data provided by domain knowledge. As one of the primary advantages, our learning framework is universal, orthogonal to both DRL models and networking applications.

A neural network defines a conditional probability parameterized by weights θ that maps a certain system state to an action. The DRL training updates θ iteratively to produce the optimal actions of training instances. The student is modeled as an MDP as described in Section II-A and the teacher provides external knowledge to the student network. For the state at which the teacher provides action advice, i.e. the state pertinent to a wrong or risky decision, the RL agent will receive an additional penalty $l(s_t, a_t, a_t')$. This new loss quantifies the difference between the student's decision and the teacher's advice. The MDP under the guidance of a teacher can be defined as $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \mathcal{D})$ with $\mathcal{D}$ containing the states where the teacher provides advice. In this setting, the objective of the student is given by

$$\begin{aligned} &J(\theta) \\ &= \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \eta\gamma^t r_t(s_t, a_t) + (1-\eta)\mathbf{1}\{s_t \in \mathcal{D}\} l(s_t, a_t, a_t')] \\ &= \mathbb{E}_{\pi_\theta} \\ &\quad \times [\eta \sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) + (1-\eta) \sum_{t=0}^{\infty} \mathbf{1}\{s_t \in \mathcal{D}\} l(s_t, a_t, a_t')] \\ &= \eta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)] + (1-\eta)\mathbb{E}_{\pi_\theta} \\ &\quad \times [\sum_{t=0}^{\infty} \mathbf{1}\{s_t \in \mathcal{D}\} l(s_t, a_t, a_t')] \\ &= \eta v_{\pi_\theta}(s) + (1-\eta) l_{\pi_\theta}(s), \end{aligned} \quad (13)$$

where $\mathbf{1}\{s_t \in \mathcal{D}\}$ is an indicator function with its value equals to 1 when a teacher provides advice at state s_t. η is a constant in $[0, 1]$ that controls the balance between maximizing the expected cumulative reward and imitating the teaching data.

The subsequent challenges are who will play the role of the teacher, and how the teacher's action can be learned by the student network. The teacher is a set of simple *white-box* logic rules that are specified by domain-specific algorithms or human engineers. In a word, he should be simple enough and provide exact *action advice* to his neural network based student. In this work, we choose the classical rule-based expert methods as the teachers. Here we present several rules for choosing an expert method: (i) white-box logic, i.e. the expert method should be explainable; (ii) simple but effective; (3) providing exact action advice to its neural network based student. Meanwhile, we do not recommend to use the algorithms based on prediction as the expert methods, since prediction algorithms themselves are a kind of learning.

The overall Teacher-Student framework is illustrated in Fig. 6. The student is the DRL system and the teacher is the domain-specific algorithm. The main body of this framework is still the interaction process between the agent and its environment. There are three key modules that help learn teaching data, including confidence check, reward shaping and prioritized experience replay. Among them, the confidence check module can help locate the states of wrong decisions and risky decisions, reward reshaping stimulates the student to learn teacher's advice, and prioritized experience replay achieves effective training when the number of teaching data is not enough. We will give more details about these modules in the following section.

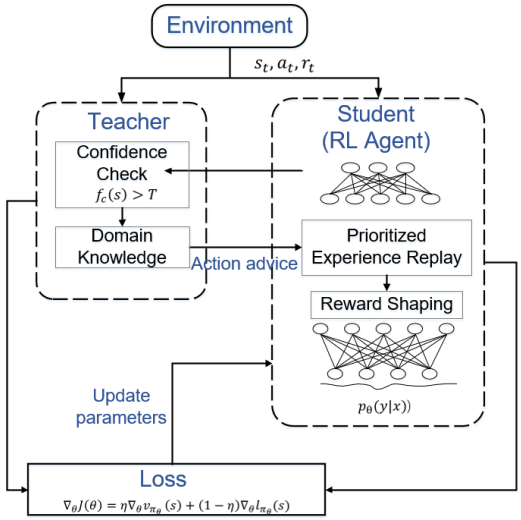


Fig. 6. Teacher-Student learning framework.

V. System Design

In this section, we describe three key components of the proposed Teacher-Student framework, including confidence check to automatically locate critical states, reward shaping and prioritized experience replay to effectively incorporate advice.

A. Confidence Check

• Challenge 1: When and how to give advice?

The training of DRL is an interactive process and each iteration would generate a large amount of trajectory data. Extracting wrong and risky decisions is undoubtedly a problem of finding a needle in a haystack. The prerequisite is to identify the critical states where DRL may not function properly.

• Solution 1: We tackle this challenge by using a process of confidence check, during which the teacher provides advice on a state if the corresponding value of the confidence metric is less than a given threshold. The domain knowledge in networking applications could help choose the confidence metric and corresponding threshold value. More specifically, the quality of experience (QoE) metric in video streaming service directly reflects a user's satisfaction of video perception. Since it is a real-time system, we can set the confidence metric as a user's instantaneous reward that captures the QoE of each video chunk. For load balancing, we choose two confidence metrics. One is the ratio of the incoming job size to the size of the server's queue and the other is based on the estimated completion time, which will be introduced in detail in section VI. However, as illustrated in [42], the expert knowledge is heuristic and may be inaccurate occasionally. Hence, this knowledge should be scrutinized before being provided to the DRL agent. Our intuition is that valid teaching advice a' should lead to a higher cumulative reward in the current learning sequence when compared to the student's intended action a. We called a critical state s if

$$f_c(s) < T, R(s, a') > R(s, a),$$

where $f_c(\cdot)$ is a function for calculating the value of the confidence metric at a given state $s \in \mathcal{S}$. T is the established threshold for the corresponding confidence metric. $R(s, a)$ is the cumulative reward from taking action a at state s.

B. Reward Shaping

• Challenge 2: How can a student network learn advice?

Once the action advice is provided, our goal becomes to embed them into the student DRL agent. A naïve approach is that we directly use the action advice provided by the teacher for training. The rationality behind it is to help the agent explore the state and action space better. However, as shown in Fig. 21, our experiments prove that this method is invalid. We need a more efficient method to help the agent learn the teaching data.

• Solution 2: From Equation (13) we can get

$$\begin{aligned}\bigtriangledown_\theta J(\theta) &= \eta \bigtriangledown_\theta v_{\pi_\theta}(s) + (1-\eta) \bigtriangledown_\theta l_{\pi_\theta}(s) \\ &= \eta \mathbb{E}_{\pi_\theta}[G_t \bigtriangledown_\theta \log \pi_\theta(s_t, a_t)] + (1-\eta) \bigtriangledown_\theta l_{\pi_\theta}(s),\end{aligned} \tag{14}$$

where the last equality is because $\bigtriangledown_\theta v_{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[G_t \bigtriangledown_\theta \log \pi_\theta(s_t, a_t)]$ and $G_t = q_{\pi_\theta}(s_t, a_t)$ in [19]. In addition, G_t can also be in the form of advantage function $G_t = q_{\pi\theta}(s_t, a_t) - b(s)$ in Actor-Critic, or in the form of $G_t = \frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}(q_{\pi_\theta}(s_t, a_t) - b(s))$ in PPO.

The original Teacher-Student method in [15] sets the loss function to the KL divergence of probability distribution output by the students and that given by the teacher, in which the probability distribution determines the classification result. Similarly, for the RL agent with discrete actions, we can also repress the teacher's action advice in the form of one-hot and calculate KL divergence. The problems are that: (i) this approach is only suitable for discrete action space. Since the output of the agent with continuous action is mean and variance of a normal distribution, it is difficult for a teacher to provide accurate values; (ii) The choice of the parameter η is extremely important and sensitive if we calculate gradients of two parts in objective function separately. This becomes worse when neural networks are over-parameterized.

Reward shaping [43] is an important approach in RL whereby additional rewards are used to guide the learning agent for faster training speed. We propose a new shaping method as a variant of reward shaping. Specifically, a new updating function is used for DRL that injects networking *domain-specific rules* in the student network by modifying the value of G_t of the corresponding teaching data instead of providing additional rewards. This approach could force the student to learn teacher's advice while not impacting the update of other normal state action pairs. The new updating function of DRL agent we use in the Teacher-Student approach is:

$$\begin{aligned}\theta \leftarrow \theta + (1 - \mathbf{1}\{s_t \in \mathcal{D}\})\delta \bigtriangledown_\theta \log \pi_\theta(s_t, a_t) G_t \\ + \mathbf{1}\{s_t \in \mathcal{D}\}\delta \bigtriangledown_\theta \log \pi_\theta(s_t, a'_t) F_t,\end{aligned} \tag{15}$$

here, the expression $\bigtriangledown_\theta \log \pi_\theta(s_t, a_t)$ provides the direction that updates the policy parameter θ so as to improve $\pi_\theta(s_t, a_t)$.

Equation (15) takes a step towards this goal and the step size also depends on how large the G_t and F_t are. The shaping function $F : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^+$ is a real-valued function. At a teaching state s_t, $F_t \propto (R' - R)$ where R represents the cumulative reward gained by performing student intended action, and R' is by using the teacher's advised action with $R' \geq R$. Note that in some special systems, we may not be able to obtain R' all the time. For example, obtaining R' requires a step backward of the corresponding environment. This can be achieved in the simulation but not in a real system. In addition, for applications like job scheduling, such queuing systems exist the problem of delayed reward, i.e. a single-step reward signal does not reflect the improvement of the system performance with respect to the teaching action, and hence the accurate R' is unknown. So we recommend choosing this form F_t in streaming systems, since the instantaneous reward can reflect the true performance of an action to some extent. In other systems, it is chosen based on experience. Nevertheless, the value of F_t is much more robust than the value of η.

C. Prioritized Experience Replay

• Challenge 3: How to train the student network with little teaching data effectively?

The amount of teaching data varies considerably across different networking applications and different confidence metrics. We need a more effective method to train the student model with little teaching data available.

• Solution 3: We employ a prioritized experience replay method in [44] to address this challenge. Experience replay enables RL agents to remember and reuse the history data. Authors in [44] show that prioritizing experience replay data will make the learning more efficient. Since different transitions (i.e. the state and advised action pairs) are not equally useful, we therefore introduce the prioritized experience replay mechanism whose kernel component is the criterion by which the importance of each transition is measured. The improved cumulative reward value is chosen to measure the importance of different state-action pairs. Note that our approach has a similar function with the TD-error [44] while the latter is not suitable for policy-based training algorithms which are most used by networking applications. In addition, the stochastic method is implemented to ensure a non-zero probability of sampling low priority transitions. Formally, the probability of sampling transition i is given by:

$$P_i = \frac{p_i^\alpha}{\sum_i p_i^\alpha}, \tag{16}$$

where α is a positive exponent regarding the weight of priority. Here, $p_i \in (0, 1]$ is the priority of transition i calculated by $p_i = \frac{1}{rank(i)}$. Here, $rank(i)$ is the order of transition i among all the state-action pairs that are sorted ascendingly by their importance on improving the cumulative reward value $(R' - R)$.

The proposed method is suitable for (i) policy gradient based RL algorithms, (ii) the application where domain knowledge can provide exact action advice and (iii) the domain knowledge or the expert method is capable of solving the robustness problem that we are interested in.

Algorithm 1 Leveraging Domain Knowledge for Robust DRL

Input: Pre-trained DNN, domain-specific Rules
Output: Policy π_θ

Initialize teacher replay buffer $\mathcal{H}$, student replay buffer $\mathcal{U}$
for each iteration **do**
  $\Delta\theta \leftarrow 0$
  for each episode **do**
    Obtain $s_t, a_t, f_c(s_t)$
    if $f_c(s_t) < T$ **then**
      Get valid action advice a'_t from teacher
      Store transition (s_t, a'_t, r_t) in $\mathcal{H}$
    else
      Store transition (s_t, a_t, r_t) in $\mathcal{U}$
    end if
  end for
  Sampling teaching data $\mathcal{H}' \sim P(i) = p_i^\alpha / \sum_i p_i^\alpha$, $p_i = 1/rank(i)$
  for each training data **do**
    Compute weight change with shaping
    $\Delta\theta \leftarrow (1 - \mathbf{1}_{\mathcal{H}'}(s_t))\delta \bigtriangledown_\theta \log \pi_\theta(s_t, a_t) G_t + \mathbf{1}_{\mathcal{H}'}(s_t)\delta \bigtriangledown_\theta \log \pi_\theta(s_t, a'_t) F_t$
  end for
  Update weights $\theta \leftarrow \theta + \Delta\theta$
end for

D. Training

The training process is carried out in two stages, the scratching and teaching phases. In the scratching phase, the student interacts with the environment, continually generating training data and using it to improve strategy iteratively, as the standard training process of DRL. When the teaching phase starts, the Algorithm 1 shows the pseudocode of teaching phase, the teacher supervises the student performance and provides advice in the form of an exact action decision when the value of confidence metric $f_c(s_t)$ is less than a threshold. Then the valid teacher advice data would be stored in the teacher replay buffer. The training data also contains two parts, including student experience data that is collected through interacting and teaching data that is sampled from teaching replay buffer based on the priorities, which are calculated by how much valuable of each action advice. Note that the teacher only participates in teaching phase, part of training process, for the reasons: (i) reducing teaching cost, (ii) failing to explore the action space effectively, thus falling into a local optimum if integrating teacher's advice data when the student policy network is not mature. The iterations continue until the "student" converges to a DNN that strikes a delicate balance between the expected cumulative reward optimization and imitating the behavior of a domain-specific logic. This process makes the neural network more reliable while consistently performing better than domain-specific algorithms on various environment settings.

VI. Performance Evaluation

A. Experimental Setup

We implement the proposed Teacher-Student framework in three representative networking applications using DRL: (i) Pensieve [6] for dynamic adaptive streaming over HTTP; (ii) DeepLB for load balancing and (iii) Aurora [5] for TCP congestion control. Unless specified explicitly, the parameter configuration is in line with that of each original work.

1) Video Streaming: Consider a video with a length of 193 seconds that is encoded by H.264 codec at the set of bitrates $\{300, 750, 1200, 1850, 2850, 4300\}$Kbps, and is segmented into 48 chunks. We conduct experiments in both simulated system and real system. In the simulated system, the synthetic traces are generated by using a Markov model in which each state is an average throughput ranging from 0.3 Mbps to 4.3 Mbps at the step size of 0.4 Mbps. The transition probability between different states follows a geometric distribution. Each throughput value is drawn from a Gaussian distribution parameterized with the current average throughput and the uniformly distributed variance between 0.05 and 0.5. In addition to the synthetic traces, we also use three real-world datasets, including:

- FCC18. The bandwidth measurement results of broadband network in 2018 published by FCC.
- Norway. A 3G/HSDPA mobile dataset generated using mobile devices in Norway that were streaming video via bus, train, etc.
- HSR. A 4G bandwidth measurement results on high-speed rails in 2018 with fluctuating bandwidths.

We directly use the further cleaned version of the above datasets, which is provided by [6] and [26]. The QoE metric, based on which the system performance is assessed, is calculated by:

$$QoE = \sum_n q(R_n) - \mu \sum_n T_n - \sum_n |q(R_{n+1} - q(R_n)|, \tag{17}$$

where $q(R_n)$ specifies video quality of chunk n, T_n is the rebuffering time with μ the corresponding penalty, and the last term $|q(R_{n+1} - q(R_n)|$ ensures playback smoothness by providing penalty on changes of video quality. Since the QoE metric is relatively subjective, we choose two forms of $q(R_n)$ in our following experiments. Specifically,

- QoE_{lin} : $q(R_n) = R_n$, R_n is the bitrate of chunk n, used by MPC.
- QoE_{log} : $q(R_n) = \log(R_n/R_{min})$, R_{min} represents the minimum bitrate of video chunks. QoE_{log} implies a diminishing marginal effect of user experience on the video chunk bitrate.

Real System: We implement the Pensieve model and our model in real-world dash.js system. Tensorflow.js is used to store and load the trained model. In our setup, we run a virtual machine on our notebook as the client and a host machine as an IIS web server. The client keeps sending XMLHttpRequests to the server to download the video chunks. The client video player is a Google Chrome browser. We convert the trace files to .bat files and use Dummynet at the client side to control the network bandwidth so that a time varying bandwidth can be emulated with realistic bandwidth traces. The video files in our testing are directly from Pensieve, i.e. the same video durations, bitrates and chunk sizes.

The ***teacher*** we use for Pensieve is the classical buffer-based algorithm (BBA) proposed in [16]. BBA defines a buffer size that is the difference between the downloading and the playback progresses. A larger buffer size means a safer situation against playback interruption. BBA selects the lowest bitrate if the current buffer size is below 5s, selects the highest bitrate if it is above 35s, and controls the requested bitrate linearly if it is in between. Since BBA is widely adopted and well-performed, it is selected as the teacher to guide a more robust student neural network. Other ABR algorithms, such as MPC [45] and BOLA [46], are essentially learning based or prediction based algorithms, which are not suitable for the teacher required for simple decision logic. The confidence metric of the teacher is Pensieve's instantaneous reward that reflects the QoE of the last downloaded video chunk.

2) Load Balancing: In our experiment, the arrival of jobs is a Poisson process with the mean inter-arrival time of 70 seconds. We consider the case of two types of jobs, the small jobs and the large jobs. The request processing time of small jobs is uniformed distributed between $20 \sim 50$ units, and that of large jobs is also uniformed distributed but in the range between $200 \sim 300$ units. For each incoming job, it is a small one with probability 0.8 and is a large one with probability 0.2. We use Actor-Critic algorithm for RL training. The actor network is a three-layer fully connected neural network. The number of neurons from the input layer to the output layer is 3, 200, 128, 2 respectively. The output of action network is the probability distribution that allocates an incoming job to the candidate servers. The critic network has the same structure as the action network, except that the number of neuron in the output layer is 1. The output of critic network is an estimate of the value function of the input state.

Two confidence metrics are selected in load balancing, including:

- Confidence Metric A: The ratio of the incoming job size to the size of server's queue.
- Confidence Metric B: $j + min(q_1, q_2, \cdots, q_k) - \lambda$, where j is the size of the incoming job, q_i represents the queue length of server i, and λ is the average arrival interval of consecutive jobs.

Recall that the number of confidence values corresponding to metric A is the same as the number of servers in the system. Each time the RL agent chooses an action, the confidence values are calculated with respect to each server. The teacher provides action advice when the difference between the confidence value of the selected server and the maximum confidence value among all servers exceeds the given threshold. We choose the shortest processing algorithm (SP) as the ***teacher*** in load balancing application. SP calculates the total length of each server queue, including the total size of jobs waiting in the queue and the remaining size of the job currently being processed, and schedules the incoming job to the server corresponding to the shortest queue length.

3) TCP Congestion Control: We implement TCP Aurora in network simulator 3 (NS3) [47] that faithfully complies with the emulation in Pantheon. The TCP DRL agent is retrained using OpenAI Gym for NS3. We adopt a well-trained model of TCP Aurora to be the start point of our retraining. This model is trained using an event-driven simulator because the interaction between the sender and the environment in the real world is time-consuming, much larger than the computation of gradients in the neural network. The simulator produces single link traces with bandwidth between 100 packets per second to 500 packets per second, round-trip propagation delay between 100 ms and 1000 ms, queue size between 2 packets and 2981 packets and loss rate between 0 and 5%. The sender is given an initial pacing rate between 30% and 150% of the link bandwidth. The DNN we used here has 2 hidden layers, with 32 neurons in each layer. The evaluation of the pre-trained and retrained TCP Aurora is carried out in NS3.

We use the classical BBR algorithm as the ***teacher*** whose wisdom is to let the sending rate match the bandwidth delay product (BDP). Though BBR is expected to operate at the optimal usage of network bandwidth, it heavily relies on the estimation of BDP. We hereby argue that the BDP estimate is not an accurate value to tune the TCP sending rate, but can tell whether the current sending rate is obviously wrong or not. By reviewing the sequence of the states as well as Aurora's actions, we observe that Aurora is sometimes too conservative when the bandwidth utilization is low, and is still aggressive when the network is already congested. We set up two thresholds, α_{min} and α_{max}. If the ratio of the current sending rate over the available bandwidth is below α_{min} or above α_{max}, Aurora will be endowed with an action suggested by BBR. The choice of α_{min} and α_{max} is guided through human experience. In our setting, we set α_{min} to 0.6 and α_{max} to 1.2, and actually they are rather robust to a wide range of configurations.

B. Experimental Results

1) Video Streaming: Since the real video playback is extremely time-consuming (about 10 days for training a converged model), we train the original Pensieve model in the simulated system and test it in the real-world system. A well-trained version of Pensieve is selected and we retrain it by using teaching data provided by its BBA teacher. The instantaneous reward in each single step can serve to indicate the rough quality of the agent's decision. Therefore, we locate the risky decisions based on the QoE value of the downloaded video chunk where the confidence threshold is -20. Fig. 7 illustrates the buffer size, requested bitrates, cumulative rebuffering time, cumulative reward and throughput of the original Pensieve (red solid lines) and the teacher guided Pensieve-TE (blue dash lines) from top to bottom that evolves. One can observe that the original Pensieve encounters a rebuffering around 5 seconds at time 73s, as shown in the area between the two green vertical lines. Specifically, Pensieve requires a video chunk with bitrate 750Kbps even if the current buffer size is relatively small, resulting a rebuffer, suddenly dropped reward (QoE) and poor throughput value. With the actions advised by BBA, the teacher guided Pensieve-TE requires a lower bitrate and avoids such a playback interruption.

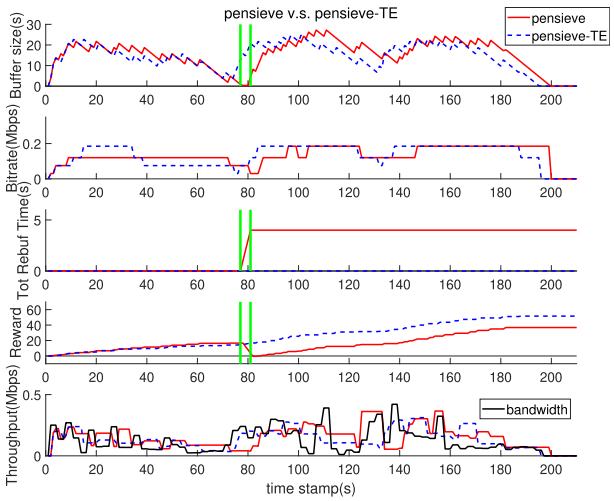


Fig. 7. Comparison of Pensieve and Pensieve-TE.

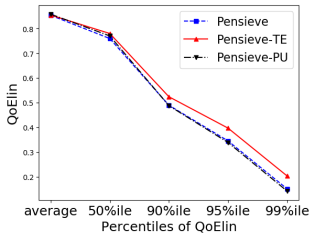


Fig. 8. Percentiles of QoE values on synthetic dataset.

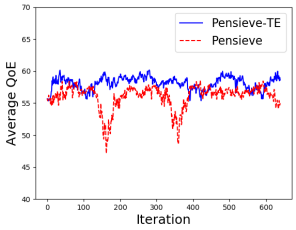


Fig. 9. Average performance during retraining.

Although Pensieve and Pensieve-TE have very close average QoE values, the risky decisions made by Pensieve might have caused poor tail performance. Fig. 8 compares their tail QoE_{lin} values in real system 300 times, where x-coordinate represents different percentiles of the tails, and y-coordinate displays the normalized average QoE_{lin} values of these tails. Obviously, Pensieve-TE exhibits a much better tail performance than Pensieve, especially at the 95th and 99th percentiles. Specifically, the proposed method improves the 90%ile, 95%ile and 99%ile performance of QoE_{lin} by 7.6%, 8.8% and 10.7% respectively. We further conduct a set of experiments to justify the causality that the improved QoE of Pensieve-TE comes from the Teacher-Student framework, rather than the insufficient training of the original Pensieve. Fig. 9 shows that Pensieve-TE does not sacrifice the average QoE_{lin} during the retraining epochs, compared with the

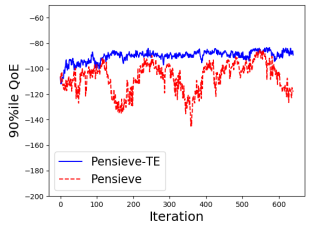


Fig. 10. 90th percentile reward of Pensieve and Pensieve-TE.

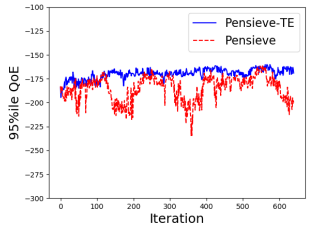


Fig. 11. 95th percentile reward of Pensieve and Pensieve-TE.

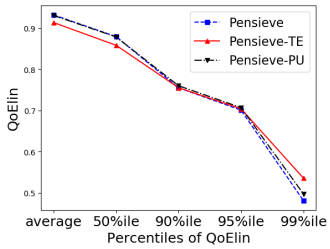


Fig. 12. Percentiles of QoElin values on FCC18.

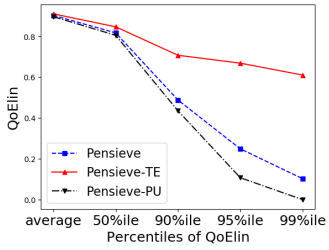


Fig. 13. Percentiles of QoElin values on HSR.

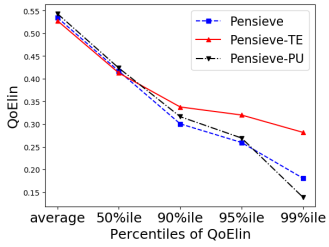


Fig. 14. Percentiles of QoElin values on Norway.

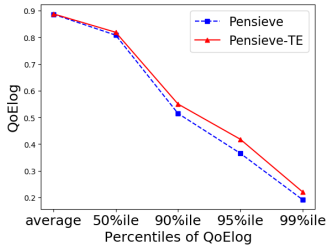


Fig. 15. Percentiles of QoElog values on synthetic dataset.

original Pensieve. Fig. 10 and Fig. 11 demonstrate the average QoE_{lin} value of the 90th and 95th percentiles during the retraining. We observe that the tail performance of Pensieve-TE gradually improves till convergence, and it is rather stable. Fig. 12 compares tail QoE_{lin} values on FCC18 dataset, it seems that Pensieve-TE does not perform well at first glance. The reason is that FCC18 is a dataset with relatively large bandwidth, i.e. pensieve is able to request video chunks with high bitrate while keeping the buffer sufficient, hence the BBA teacher is not necessary in this circumstance. However, in Fig. 13 and Fig. 14 with relatively lower bandwidth, the Pensieve-TE obviously improves tail QoE_{lin} values. As for the QoE_{log} metric, Fig. 15 - Fig. 18 demonstrate the performance of Pensieve and Pensive-TE on four datasets, respectively. Similarly, the Pensieve-TE achieves better tail performance overall, especially on Norway dataset, where even average performance has a significant improvement. Note that the preference to tail or average performance depends on concrete environment. For example, when broadcasting important meetings, we pay more attention to tail. While for daily watching video, we care more about average performance. The trade-off between different metrics needs to consider the specific environment.

We also conduct experiments to compare the Teacher-Student method (Pensieve-TE) with punishment reward method (Pensieve-PU). In the punishment reward method, we use the same confidence metric as Pensieve-TE to detect wrong/risky decisions. Instead of providing teaching advice and executing reward shaping afterward, Pensieve-PU punishes the DRL agent with a large negative reward at critical states. Pensieve-PU is tested with QoElin metric across four datasets used in this paper. Fig. 8, Fig. 12 and Fig. 14 manifest that Pensieve-PU achieves nearly similar tail performance with the original Pensieve. However, in Fig. 13, Pensieve-PU is even inferior to Pensieve. We speculate that the punishment reward might not be able to guide the DRL agent to the action advised by the domain knowledge teacher, thus failing to improve robustness. In fact, solely relying on the punishment reward or the action advised through the domain knowledge does not help the DRL agent to learn the appropriate action. A synthetic approach of reward sharing and action advising is thus preferred.

In Pensieve-TE, the confidence threshold is a hyper parameter. We next evaluate the sensitivity of Pensieve-TE on the confidence threshold by modifying its value to -10, -15 and -20 respectively, as shown in Fig. 19. The teachers' advice based on three different confidence thresholds are different, but their outcomes on the tail performance remain to be very close. Hence, we can reasonably claim that the Teacher-Student framework is robust to the setting of the confidence threshold in Pensieve.

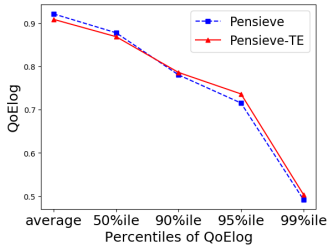


Fig. 16. Percentiles of QoElog values on FCC18.

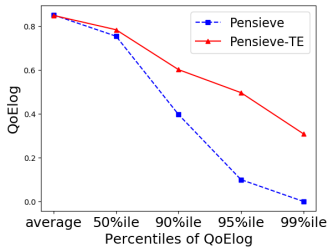


Fig. 17. Percentiles of QoElog values on HSR.

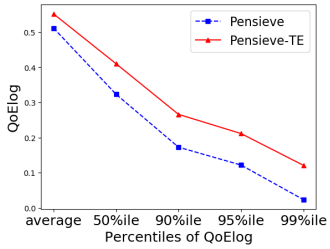


Fig. 18. Percentiles of QoElog values on Norway.

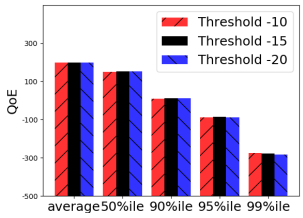


Fig. 19. Using different threshold of confidence value.

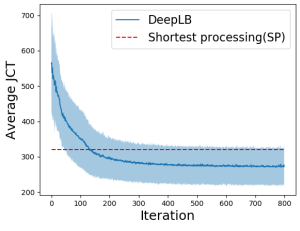


Fig. 20. DeepLB training performance vs training epochs.

2) Load Balancing: We first train a DeepLB agent from scratch and select a converged model for subsequent enhancement. Fig. 20 shows that as the number of iterations increases, DeepLB outperforms the heuristic algorithm gradually, which manifests the rationality of using DRL for load balancing.

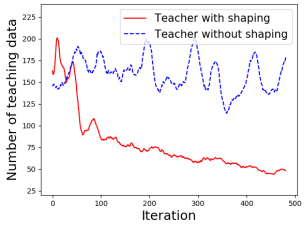


Fig. 21. The number of teaching data w and w/o reward shaping.

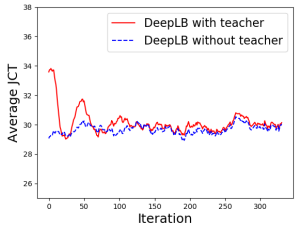


Fig. 22. DeepLB-TE retraining performance.

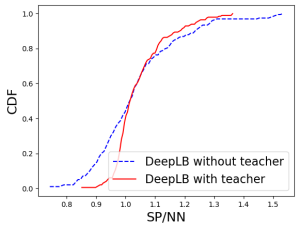


Fig. 23. DeepLB-TE vs. SP on confidence metric A.

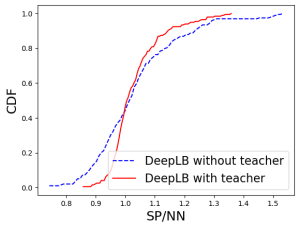


Fig. 24. DeepLB-TE vs. SP on confidence metric B.

Note that the shaded area in Fig. 20 describes different test sequences, not the large variance of model performance. We denote by DeepLB-TE the teacher guided model using our Teacher-Student framework. As shown in Fig. 21, x-coordinate is the number of re-training iterations, and y-coordinate is the number of states that the teacher needs to provide advice. The red real curve is for the retrained agent with reward shaping, and the blue dash curve is for that without reward shaping. It is thus highlighted that as the training iteration moves on, the DeepLB-TE with reward shaping relies less and less on the teaching data, but the agent without reward shaping still requires the similar amount of teaching data. This set of experiments implies that the reward shaping is essential to enable the RL agent to learn the advised actions, and to update its own policy network accordingly. Besides, Fig. 22 describes that DeepLB-TE maintains the comparable average job processing time as the original DeepLB.

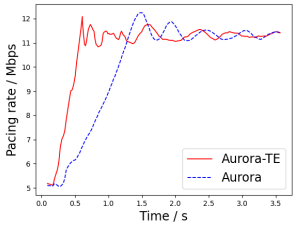


Fig. 25. Pacing rates of Aurora vs. Aurora-TE.

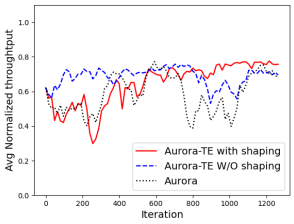


Fig. 26. Normalized TCP throughput.

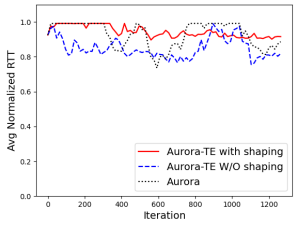


Fig. 27. Normalized TCP RTT.

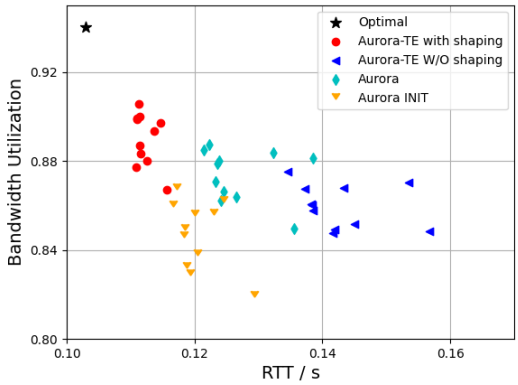


Fig. 28. Comparison of different versions of Aurora.

The Teacher-Student framework improves the robustness of the original DeepLB system. Specifically, we test 200 job sequences and each sequence consists of 70 jobs. We compare the RL agent with the shortest processing time algorithm (SP) that serves as a teacher to provide action advice at critical states. The original DeepLB, though more performant than the rule-based SP in terms of the average job processing time, is found that 42% of sequences have worse performance than the rule-based SP. This indicates that the job processing times in DeepLB have a large performance variance. Fig. 23 shows the CDF curves of the performance of DeepLB and DeepLB-TE that are normalized by the performance of SP, under the guidance of confidence metric A. With the guidance of the teacher, DeepLB-TE successfully learns the teacher's advice and yields a smaller variance of job processing times. The proposed method reduces the performance variance of DeepLB by 37% when compared with SP. Hence, DeepLB-TE is believed to be more robust and reliable than DeepLB and SP. The similar results appear on confidence metric B, as shown in Fig. 24.

3) TCP Congestion Control: We take a pre-trained TCP Aurora model as the start point and retrain it using our Teacher-Student framework in NS3. Fig. 25 shows the different pacing rate behavior of retrained model Aurora-TE and the initial start point model Aurora. We create a single link with 20 Mbps bandwidth and 40 ms round-trip propagation delay. A UDP background flow with sending rate of 8Mbps is always active. We set the initial pacing rate to 5Mbps for both models. Experiments show that Aurora needs about 1.5 seconds to catch up to the optimal pacing rate. But after retraining with intervention of BBR, Aurora-TE only takes 0.7 seconds, which is nearly 2x faster. This improvement can help the TCP flow utilizes bandwidth more efficiently, especially in the link with dynamically changing bandwidth.

To evaluate whether our approach will harm the average performance of TCP Aurora, we retrain three Aurora models using different retraining methods. After every 16 iterations, for each model we run 55 traces with the same link parameters as they were in retraining setting. Since the bandwidth and latency of every trace are different, we use the normalized throughput and the normalized round trip time (RTT) as defined below to show the performance of models on each trace. The normalized throughput is computed as the throughput divided by the link bandwidth, and the normalized RTT is computed as the fixed round trip propagation delay divided by the RTT. They both should be smaller than 1, and the bigger value means better performance.

Fig. 26 and Fig. 27 illustrate the average normalized throughput and average normalized RTT of three Aurora models as the training iteration goes on. Aurora-TE with shaping is retrained in our standard Teacher-Student approach. Aurora-TE W/O shaping is also retrained in our Teacher-Student approach but without Reward Shaping. Aurora is just retrained using NS3 traces without teacher's advice. Aurora-TE with shaping achieves a more stable performance compared to the start point model. We also find that Aurora-TE W/O shaping behaves unstable and exhibits a worse RTT performance. It signifies the importance of reward shaping in our Teacher-Student approach.

Finally, to evaluate the scalability and robustness of our retrained Aurora-TE, we consider a dynamic link whose bandwidth varies to a new value randomly chosen between 10 Mbps and 20 Mbps every 5 seconds. The link has a fixed round-trip delay of 100ms. We run 11 times for each model and each run lasts 20 seconds. The comprehensive results are shown in Fig. 28. Aurora INIT represents the start point model or the pre-trained model without any retraining. As expected, these models have different trade-offs between RTT and bandwidth

utilization. Among them, Aurora-TE with shaping achieves the best performance with 5% higher throughput and 6% lower RTT than Aurora INIT.

VII. Discussion

In this section, we discuss some open problems about the proposed Teacher-Student framework.

A. Why Use Teacher-Student Framework Instead of Reconsidering Reward Design Directly?

Though modifying reward is indeed a direct method, reward design cannot eliminate the wrong and risky decisions. For the wrong decision, take Pensieve, the video streaming system selected in this paper, as an example. We aim at maximizing the QoE (overall performance or average performance) of a video session. The overall QoE of a video session can be expressed as the sum of multiple QoEs of video chunks, and the QoE of a single video chunk can be directly calculated when the download is completed. It is natural that we use the QoE of a downloaded video chunk as the instantaneous reward. Such reward design is simple, direct, and accurate, but the experimental results still show the existence of wrong decisions, as shown in Fig. 7. The reason is that the policy of the Deep RL agent is essentially a neural network, which is an approximation of the table strategy with inherent approximation errors. Furthermore, the neural network itself also has the typical problem of adversarial samples, which can also lead to wrong decisions. Such problems can not be solved by reward design. Therefore, we are more inclined to use domain knowledge to locate and correct these abnormal decisions.

The main problem solved in this paper is how to design a general framework to integrate domain knowledge into DRL-based agents to solve the problem of wrong and risky decisions in neural network based policies. In the proposed Teacher-Student framework, we only need domain knowledge as a teacher to give action suggestions. However, if domain knowledge is transformed into a crafted reward design, it becomes a case-specific solution, which is not highly versatile.

B. Why Not Integrate Average Performance, Tail Performance and Performance Variance Into a Single Metric?

Performance metrics like variance or tail performance pose a great challenge to single-step reward design. For average performance, taking load balancing as an example, we can set the reward to $r_{aver}(s_t, a_t) = -(t_i - t_{i-1}) \times n$, where n represents the number of active jobs in the system during period $[t_{i-1}, t_i]$. Both the arrival and completion of jobs will trigger a state transition. Suppose a load balancing agent needs to allocate a sequence of N jobs, and the episode ends at time step M, then $T = \sum_{t=0}^{M} r_{aver}(s_t, a_t)$ is the overall completion time of all the jobs in the sequence. Then T/N is the average performance that we aim to optimize. However, the variance is calculated in the unit of an episode which may include hundreds or thousands of state-action pairs, making it difficult to define the proper single-step reward $r_{var}(s_t, a_t)$. We can not get the variance through the accumulation of single-step rewards. This problem also exists in optimizing tail performance. Research works related to this topic often design case-specific updating algorithms, which are opaque to different applications. The proposed Teacher-Student Framework may not be the best, while it is feasible and can be applied to various networking applications.

If we define the goal as the weighted sum of average performance, tail performance and performance variance, which is formally $\alpha R_{aver} + \beta R_{tail} + \gamma R_{var}$. The optimization result is likely towards a specific metric due to the unknown fundamental complex trade-off between different metrics. In addition, different networking systems, even the same networking system under different environments, have different preferences on performance metrics. Thus choosing the weights of different metrics becomes a case-specific problem. Based on the above considerations, we recommend using the proposed Teacher-Student method.

VIII. Conclusion

In this paper, we shed light on how the domain knowledge in networking is leveraged to improve robustness of DRL-based learning systems. We model these systems based on input-driven MDP, provide theoretical analysis on the system performance influenced by wrong and risky decisions. A Teacher-Student learning framework is proposed in which a domain-specific algorithm serves as the teacher to provide advice to the student neural network. The evaluation results show that: (1) the proposed method is suitable for both the discrete and continuous action space; (2) the proposed method successfully reduce the performance variance, improve tail performance and enable a more stable DRL agent while not sacrificing the average performance; (3) reward shaping plays an important role when incorporating the teaching data into student network, i.e. directly using action advice of teacher cannot improve the robustness of student network; (4) the proposed method is robust to different hyperparameter settings.

References

[1] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[2] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[3] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.
[4] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 270–288.
[5] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 3050–3059.
[6] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 197–210.
[7] Y. Guan, Y. Zhang, B. Wang, K. Bian, X. Xiong, and L. Song, "PERM: Neural adaptive video streaming with multi-path transmission," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 1103–1112.

[8] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 1967–1976.

[9] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "QARC: Video quality aware rate control for real-time video streaming based on deep reinforcement learning," in *Proc. 26th ACM Int. Conf. Multimedia*, Oct. 2018, pp. 1208–1216.

[10] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proc. 16th ACM Workshop Hot Topics Networks (HotNets)*, 2017, pp. 185–191.

[11] X. You, X. Li, Y. Xu, H. Feng, J. Zhao, and H. Yan, "Toward packet routing with fully-distributed multi-agent deep reinforcement learning," 2019, *arXiv:1905.03494*.

[12] Y. Zheng, Z. Liu, X. You, Y. Xu, and J. Jiang, "Demystifying deep learning in networking," in *Proc. 2nd Asia–Pacific Workshop Netw. (APNet)*, 2018, pp. 1–7.

[13] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[14] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," 2018, *arXiv:1807.02264*.

[15] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing, "Harnessing deep neural networks with logic rules," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics (Long Papers)*, vol. 1, 2016, pp. 2410–2420.

[16] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM Conf. SIGCOMM*, Aug. 2014, pp. 187–198.

[17] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Netw. Congestion*, vol. 14, pp. 20–53, Dec. 2016.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[19] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2000, pp. 1057–1063.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[21] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," 2015, *arXiv:1509.06461*.

[22] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 1928–1937.

[23] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operat. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[24] Y. Kazak, C. Barrett, G. Katz, and M. Schapira, "Verifying deep-RL-driven systems," in *Proc. Workshop Netw. Meets AI ML (NetAI)*, 2019, pp. 83–89.

[25] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 676–685.

[26] Z. Meng *et al.*, "Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 723–736, Apr. 2021.

[27] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, 2020, pp. 154–171.

[28] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, 2020, pp. 632–647.

[29] H. Mao, M. Schwarzkopf, H. He, and M. Alizadeh, "Towards safe online reinforcement learning in computer systems," in *Proc. NeurIPS Mach. Learn. Syst. Workshop*, 2019.

[30] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[31] R. Howard and J. Matheson, "Risk-sensitive Markov decision processes," *Manage. Sci.*, vol. 18, no. 7, pp. 356–369, 1972.

[32] K.-J. Chung and M. J. Sobel, "Discounted MDP's: Distribution functions and exponential utility maximization," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 49–62, Jan. 1987.

[33] L. Xia, "Mean–variance optimization of discrete time discounted Markov decision processes," *Automatica*, vol. 88, pp. 76–82, Feb. 2018.

[34] E. Altman, *Constrained Markov Decision Processes*, vol. 7. Boca Raton, FL, USA: CRC Press, 1999.

[35] D. Di Castro, A. Tamar, and S. Mannor, "Policy gradients with variance related risk criteria," 2012, *arXiv:1206.6404*.

[36] F. Maire and V. Bulitko, "Apprenticeship learning for initial value functions in reinforcement learning," in *Planning and Learning in a Priori Unknown or Dynamic Domains*, p. 23, 2005.

[37] Y. Song, Y.-B. Li, C.-H. Li, and G.-F. Zhang, "An efficient initialization approach of Q-learning for mobile robots," *Int. J. Control, Autom. Syst.*, vol. 10, no. 1, pp. 166–172, Feb. 2012.

[38] J. A. Clouse, *On Integrating Apprentice Learning and Reinforcement Learning*. Amherst, MA, USA: Univ. Massachusetts Amherst, 1996.

[39] J. Garcia and F. Fernandez, "Safe exploration of state and action spaces in reinforcement learning," *J. Artif. Intell. Res.*, vol. 45, pp. 515–564, Dec. 2012.

[40] F. Benito, "Calculating the variance in Markov-processes with random reward," *Trabajos de Estadistica Y de Investigacion Operativa*, vol. 33, no. 3, pp. 73–85, Oct. 1982.

[41] T. Fujimoto and R. Ranade, "Two characterizations of inverse-positive matrices: The Hawkins–Simon condition and the Le Chatelier–Braun principle," *Electron. J. Linear Algebra*, vol. 11, pp. 59–65, Jan. 2004.

[42] K. Efthymiadis and D. Kudenko, "Knowledge revision for reinforcement learning with abstract MDPs," in *Proc. Int. Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2015, pp. 763–770.

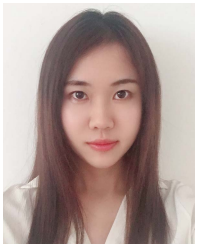
[43] A. Y. Ng *et al.*, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. ICML*, vol. 99, Jun. 1999, pp. 278–287.

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[45] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 325–338.

[46] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1698–1711, Aug. 2020.

[47] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.



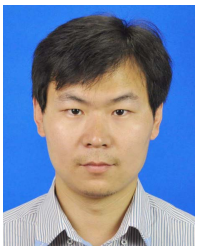
Ying Zheng (Student Member, IEEE) received the bachelor's degree from Northeastern University, China, in 2018. She is currently pursuing the Ph.D. degree with the School of Computer Science, Fudan University. Her research interests include machine learning algorithms, mobile edge computing, and networked systems.

Lixiang Lin is currently pursuing the master's degree with the Department of Electronic Engineering, Fudan University. His research interests include video streaming and reinforcement learning.

Tianqi Zhang is currently pursuing the bachelor's degree with the Department of Mathematics and Applied Mathematics, Fudan University. His research interests include multi-armed bandits, reinforcement learning algorithms, and their applications in computer networks.

Haoyu Chen received the bachelor's degree from the Department of Electrical Engineering, Fudan University, where he is currently pursuing the Ph.D. degree with the School of Computer Science. His research interests include networked systems and distributed machine learning.

Qingyang Duan received the bachelor's degree from the School of Information Science and Technology, Fudan University, in 2020, where he is currently pursuing the master's degree. His research interests include computer network systems and distributed machine learning.



Yuedong Xu received the B.S. degree from Anhui University, the M.S. degree from the Huazhong University of Science and Technology, and the Ph.D. degree from The Chinese University of Hong Kong. From 2009 to 2012, he was a Post-Doctoral Researcher with INRIA Sophia Antipolis and Université d' Avignon, France. He is currently a Tenured Associate Professor with the School of Information Science and Technology, Fudan University, China. He has published nearly 20 conferences and journals papers in premium vents, such as CoNEXT, Mobisys, Mobihoc, Infocom, and IEEE/ACM ToN. His research interests include performance evaluation, optimization, machine learning, economic analysis of communication networks, and mobile computing.

Xin Wang received the B.S. degree in information theory and the M.S. degree in communication and electronic systems from Xidian University, China, in 1994 and 1997, respectively, and the Ph.D. degree in computer science from Shizuoka University, Japan, in 2002. In 1995 and 1998, he was involved in China's pioneering telecom-level video conferencing systems and DVB-S systems with Huawei Inc., Shenzhen, China. He is currently a Professor at Fudan University, Shanghai. His research interests include quality of network service, next-generation network architecture, mobile internet, and network coding. He is a member of CCF and ACM.