THE UNIVERSITY OF CHICAGO

OPTIMIZING LEARNED NETWORKING RATE ADAPTATION VIA GUIDED REWARD REWEIGHTING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
ZHENGXU XIA

CHICAGO, ILLINOIS
DECEMBER 2024

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

It is a profound privilege to achieve a Ph.D., an endeavor that has been made possible through the support and guidance of many.

Foremost, I extend my deepest gratitude to my advisor, Prof. Junchen Jiang, whose unwavering patience and invaluable guidance have been pivotal throughout my research journey. His mentorship has been instrumental in teaching me not only the intricacies of conceiving and executing research ideas but also the art of presenting them compellingly. His influence will undoubtedly resonate throughout my professional life.

I am indebted to Dr. Francis Y. Yan, whose remarkable ability to navigate research problems has profoundly shaped my approach to formulating research questions and authoring papers. His insightful feedback has bolstered my confidence in my work, and for that, I am immensely grateful.

My sincere thanks go to Prof. Nick Feamster for serving on my committee and providing valuable assistance. His insightful comments have broadened my perspective, deepening my understanding of the field.

I am also grateful to Prof. Heather Zhen, Prof. Ben Zhao, Dr. Ganesh Ananthanarayanan, Prof. Yuanchao Shu, Dr. Nikolaos Karianakis, Dr. Kevin Hsieh, Dr. Paramvir Bahl, Prof. Ion Stoica, Dr. Chen Wang, and Dr. Jun Duan, for their constructive feedback and enriching discussions throughout the years.

My colleagues, Zhujun Xiao, Romil Bhardwaj, Lesley Yajie Zhou, Yitian Hao, Yihua Cheng, Hanchen Li, and Kuntai Du deserve special mention for their assistance in building prototypes, conducting experiments, and providing feedback. Their commitment and collaborative spirit have been essential to the success of all the work done.

A special acknowledgment goes to my family and friends, whose love and help have been a cornerstone throughout my PhD journey. Their endless encouragement and belief in my capabilities have been a constant source of strength and motivation during my most stressful

periods. I could not have achieved all of this without their strong support.

This PhD journey has been more than just an academic pursuit; it has been a life-changing experience that has prepared me for the future in ways I could never have anticipated. For all this and more, I am deeply grateful to everyone who has been part of my PhD journey.

ABSTRACT

Deep reinforcement learning (RL) based rate adaptation has been popular in the past few years. Unlike the handcrafted rate adaptation which requires manual effort from network domain experts to design and tune, RL-based rate adaptation has shown significant potential to self-adapt to different network conditions. However, it still suffers from two limitations: 1) poor generalizability across diverse network environments; and 2) lack of awareness of the user-perceived quality of experience (QoE).

In this thesis, we introduce a universal training framework for RL-based rate adaptation to overcome the three limitations currently faced. Although improving the RL model's generalizability across network environments and customizing an RL-based rate adaptation to inject QoE awareness and improve training efficiency are two separate goals, they can be achieved by the same training framework which makes use of networking domain knowledge to reweight the reward seen by the RL model at the training stage.

In this work, we cover the design of the universal training framework and instantiate the frame using use two kinds of network domain knowledge–rule-based baselines and video codecs–to address the limits respectively. Our experiments in simulated environments, emulated environments, and real-world network settings demonstrate that RL-based rate adaptation trained by the proposed training framework does have better generalizability across diverse network environments and can be customized to be aware of application layer QoE. Additionally, the RL training efficiency are largely improved in comparison to traditional RL training methods in network rate adaptation.

CHAPTER 1

INTRODUCTION

Because of deep reinforcement learning's (RL) ability to automatically create more adaptive controlling logics beyond the traditional hand-crafted heuristics, numerous effort has been made to apply RL to networking (i.e., rate adaptation) and has successfully showcased benefits over the traditional approaches. In spite of the strengths of RL-based approaches, their limitations also come to the public's attention. These limitations are three-fold:

Poor generalizability: Same as RL application in fields like robotics, RL-based approaches in networking face the generalizability problem that is RL model performs worse on testing network environments than on training network environments. This happens when the distribution of training network environments grows larger and when the testing network environment is out of the training distribution. Due to the amount and the diversity of real-world networks, RL generalizability problem desperately needs to be resolved or mitigated.

Lack awareness of QoE: RL-based rate adaptation design has been ignoring the diverse relationships between QoE of different video codecs and the network conditions, i.e., packet loss. One reason is that RL-based approaches are often trained via a reward computed by a linear combination of network throughput, packet delay, and packet loss rate rather than directly codec feedback. Lack awareness of video codec QoE can potentially prevent the power of video codecs, for example, the loss-resilience of neural video codec, from being fully released, causing users to experience suboptimal QoE.

Slow RL training convergence speed: To gain the characteristics and dynamics of the real-world networks, RL-based approaches are often trained online together with a safeguard policy which prevents RL's trial-and-error manner from causing QoE catastrophic degradation. Nevertheless, the safeguard policy can lead to poor RL learning efficiency (slow RL training convergence speed). This is because the safeguard policy not only interrupts the training process but limits the RL model's action space exploration.

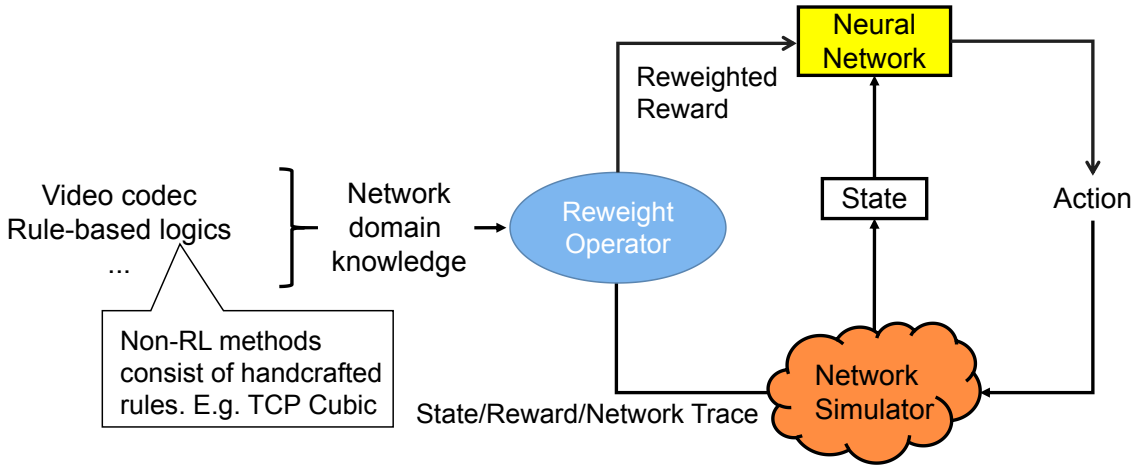


Figure 1.1: **A universal RL training framework leveraging network domain knowledge to rewweight the training reward**

In order to overcome the above limitations, this thesis proposes to resolve from the training perspective and presents a *universal RL training framework* (shown in Figure 1.1) for a networked system, i.e., rate adaptation. The core idea of the training framework to *reweight the RL training reward* guided by *network domain knowledge*. In this thesis, we instantiate this training framework using two kinds of network domain knowledge: 1) existing rule-based baselines and 2) loss-tolerant neural video codec.

GENET is an RL learning framework based on the idea of curriculum learning. Its goal is to improve RL-based solutions' generalizability across diverse network environments. It utilizes the performance gap between traditional rule-based baselines (i.e., TCP Cubic in congestion control) and RL models to automatically search for network environments which are rewarding for RL training and promotes them to training iteratively. The insight about rewarding environment is: If the current RL model performs significantly worse in a network environment than the rule-based baselines, then further training it in this environment tends to bring substantial improvement.

NVC-CC is an RL-based congestion control for real time communication, leveraging the loss tolerant characteristic of latest neural video codecs in the training process to improve its learning efficiency. By training upon reward computed from neural video codec feedback, it

removes the reliance of safeguard policies in RL training and gain awareness of video codec to reach better quality of user experience than both traditional rule-based and RL-based congestion controls for real time communication.

The rest of this dissertation is structured as follows. We first present GENET in Chapter 2 and NVC-CC in Chapter 3. Chapter 4 concludes this dissertation and discusses the future work.

CHAPTER 2

GENET: AUTOMATIC CURRICULUM GENERATION FOR LEARNING ADAPTATION IN NETWORKING

2.1 Introduction

Many recent techniques based on deep reinforcement learning (RL) are now among the state of the arts for various networking and systems adaptation problems, including congestion control (CC) [71], adaptive bitrate streaming (ABR) [101], load balancing (LB) [102], wireless resource scheduling [40], and cloud scheduling [104]. For a given distribution of training network environments (e.g., network connections with certain bandwidth patterns, delay, and queue length), RL trains a policy to optimize performance over these environments.

However, these RL-based techniques face two challenges that can ultimately impede their wide use in practice:

- **Training in a wide range of environments:** When the training distribution spans a wide variety of network environments (e.g., a large range of possible bandwidth), an RL policy may perform poorly even if tested in the environments drawn from the same distribution as training.
- **Generalization:** RL policies trained on one distribution of synthetic or trace-driven environments may have poor performance and even erroneous behavior when tested in a new distribution of environments.

Our analysis in §2.2 will reveal that, across three RL use cases in networking, these challenges can cause well-trained RL policies to perform much worse than traditional rule-based schemes in a range of settings.

These problems are not unique to networking. In other domains (e.g., robotics, gaming) where RL is widely used, it is also known that RL models have performance issues in both

new environments drawn from the training distribution and new environments drawn from an unseen distribution [138, 84, 111, 163, 112]. There have been many efforts to address these issues by enhancing offline RL training or retraining a deployed RL policy online. Since updating a deployed model is not always possible or easy (e.g., loading a new kernel module for congestion control or integrating an ABR logic into a video player), we focus on improving RL training offline.

A well-studied paradigm that underpins many recent techniques to improve RL training is *curriculum learning* [112]. Unlike traditional RL training that samples training environments in a random order, curriculum learning generates a training curriculum that gradually increases the difficulty level of training environments, resembling how humans are guided to comprehend more complex concepts. Curriculum learning has been shown to improve generalization [106, 21, 47] as well as *asymptotic* performance [146, 79], namely the final performance of a model after training runs to convergence. Following an easy-to-difficult routine allows the RL model to make steady progress and reach good performance.

In this work, we present *GENET, the first training framework that systematically introduces curriculum learning to RL-based networking algorithms.* GENET automatically generates training curricula for network adaptation policies. The challenge of curriculum learning in networking is how to sequence network environments in an order that prioritizes highly *rewarding* environments where the current RL policy's reward can be considerably improved. Unfortunately, as we show in §2.3, several seemingly natural heuristics to identify rewarding environments suffer from limitations.

- First, they use *intrinsic properties* of each environment (e.g., shorter network or workload traces [104] and smoother network conditions [58] are supposedly easier), but these intrinsic properties fail to indicate whether the current RL model can be improved in an environment.

- Second, they use *handcrafted heuristics* which may not capture all aspects of an en-

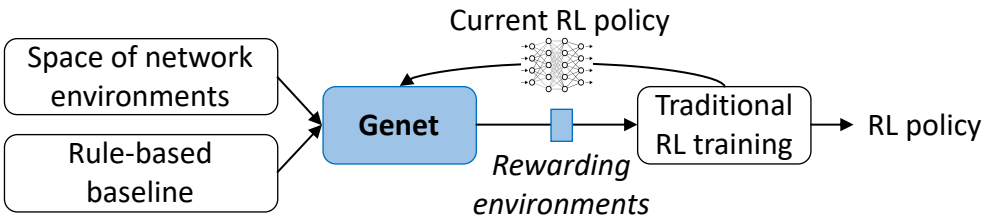


Figure 2.1: GENET creates training curricula by iteratively finding rewarding environments where the current RL policy has a large gap-to-baseline.

vironment that affect RL training (e.g., bandwidth smoothness does not capture the impact of router queue length on congestion control, or buffer length on adaptive video streaming). Each new application (e.g., load balancing) also requires a new heuristic.

The idea behind GENET is simple: An environment is considered *rewarding* if the current RL model has a large *gap-to-baseline*, i.e., how much the RL policy's performance falls behind a traditional *rule-based* baseline (e.g., Cubic or BBR for congestion control, MPC or BBA for adaptive bitrate streaming) in the environment. We show in §2.4.1 that the gap-to-baseline of an environment is highly indicative of an RL model's potential improvement in the environment. Intuitively, since the baseline already shows how to perform better in the environment, the RL model may learn to "imitate" the baseline's known rules while training in the same environment, bringing it on par with—if not better than—the baseline. On the flip side, if an environment has a small or even negative gap-to-baseline, chances are that the environment is intrinsically hard (a possible reason why the rule-based baseline performs badly), or the current RL policy already performs well and thus training on it is unlikely to improve performance by a large margin. A small gap-to-baseline might also arise when the rule-based baseline has poor performance yet the RL model still has a large room for improvement. GENET ignores this case, but we will discuss it in §2.7.

Inspired by the insight, GENET generates RL training curricula by iteratively identifying rewarding environments where the current RL model has a large gap-to-baseline and then adding them to RL training (Figure 2.1). For each RL use case, GENET parameterizes

Use case	**Observed state** (policy input)	**Action** (policy output)	**Reward** (performance)
Adaptive Bitrate (ABR) Streaming	future chunk size, history throughput, current buffer length	bitrate selected for the next video chunk	$\sum_i(\alpha\cdot\text{Rebuf}_i+\beta\cdot\text{Bitrate}_i + \gamma\cdot\text{BitrateChange}_i)/n$
Congestion Control (CC)	RTT inflation, sending/receiving rate, avg RTT in a time window, min RTT	change of sending rate in the next time window	$\sum_i(a\cdot\text{Throughput}_i + b\cdot\text{Latency}_i + c\cdot\text{LossRate}_i)/n$
Load Balancing (LB)	past throughput, current request size, number of queued requests per server	server selection for the current request	$-\sum_i\text{Delay}_i/n$

Table 2.1: RL use cases in networked systems. Default reward parameters: $\alpha = -10$ (rebuffering in seconds), $\beta = 1$ (bitrate in Mbps), $\gamma = -1$ (bitrate change in Mbps), $a = 120$ (throughput in kbps), $b = -1000$ (latency in seconds), $c = -2000$. Details in A.5.

the network environment space, allowing us to search for rewarding environments in both synthetically instantiated environments and trace-driven environments. GENET also uses Bayesian Optimization (BO) to facilitate the search in a large space. In particular, we cast the search for environments with a large gap-to-baseline as a maximum-search problem of a blackbox function in a high-dimensional space where each point represents a set of environment configurations and the function value is the gap-to-baseline. BO is then used to find a set of training environments with large gap-to-baselines.

GENET is generic, since it does not use handcrafted heuristics to measure the difficulty of a network environment; instead, it uses rule-based algorithms, which are abundant in the literature of many networking and system problems, to generate training curricula. Moreover, by focusing training on places where RL falls behind rule-based baselines, GENET directly minimizes the chance of performance regressions relative to the baselines. This is important, because system operators are more willing to deploy an RL policy if it outperforms the incumbent rule-based algorithm in production without noticeable performance regressions.[1]

1. An example of this mindset is that a new algorithm must compete with the incumbent algorithm in A/B testing before being rolled out to production.

We have implemented GENET as a separate module with a unifying abstraction that interacts with the existing codebases of RL training to iteratively select rewarding environments and promote them in the course of training. We have integrated GENET with three existing deep RL codebases in the networking area—adaptive video streaming (ABR) [8], congestion control (CC) [2], and load balancing (LB) [7].

It stands to reason that GENET is not without limitations. For instance, GENET-trained RL policies might not outperform all rule-based baselines (§2.5.5 shows that when using a naive baseline to guide GENET, the resulting RL policy could still be inferior to stronger baselines). GENET-trained RL policies may also achieve undesirable performance in environments beyond the training ranges (e.g., if we train a congestion-control algorithm on links with bandwidth between 0 and 100 Mbps, GENET will not optimize for the bandwidth of 1 Gbps). Moreover, GENET does not guarantee adversarial robustness which sometimes conflicts with the goal of generalization [119].

Using a combination of trace-driven simulation and real-world tests across three use cases (ABR, CC, LB), we show that GENET improves asymptotic performance by 8–25% for ABR, 14–24% for CC, 15% for LB, compared with traditional RL training methods. GENET aims to optimize an RL model's asymptotic performance (i.e., in-distribution generalizability), and it does not explicitly optimize the generalization in arbitrary test environments (i.e., out-of-distribution generalizability). That said, our empirical test results show that GENET-trained models improve not only asymptotic performance, but also the performance in unseen network environments.

The traces and scripts used in GENET are released at `https://github.com/GenetProject/Genet`.

2.2 Motivation

Deep reinforcement learning (RL) trains a deep neural net (DNN) as the decision-making logic (policy) and is well-suited to many sequential decision-making problems in networking [102, 62].[2] We use three use cases (summarized in Table 2.1) to make our discussion concrete:

- An **adaptive bitrate (ABR)** algorithm adapts the chunk-level video bitrate to the dynamics of throughput and playback buffer (input state) over the course of a video session. ABR policies, including RL-based ones (Pensieve [101]), choose the next chunk's bitrate (output decision) at the chunk boundary to maximize session-wide average bitrate, while minimizing rebuffering and bitrate fluctuation.

- A **congestion control (CC)** algorithm at the transport layer adapts the sending rate based on the sender's observations of the network conditions on a path (input state). An example of RL-based CC policy (Aurora [71]) makes sending rate decisions at the beginning of each interval (of length proportional to RTT), to maximize the reward (a combination of throughput, latency, and packet loss rate).

- A **load balancing (LB)** algorithm in a key-replicated distributed database reroutes each request to one of the servers (whose real-time resource utilization is unknown), based on the request arrival intervals, resource demand of past requests, and the number of outstanding requests currently assigned to each server.

We choose these use cases because they have open-source implementations (Pensieve [8] for ABR, Aurora [2] for CC, and Park [7] for LB). Our goal is to improve existing RL training

2. There are rule-based alternatives to DNN-based policies, but they are not as expressive and flexible as DNNs, which limits their performance. Oboe [20], for instance, sets optimal hyperparameters for RobustMPC based on the mean and variance of network bandwidth and as shown in §2.5.4, is a very competitive baseline, but it performs worse than the best RL strategy.

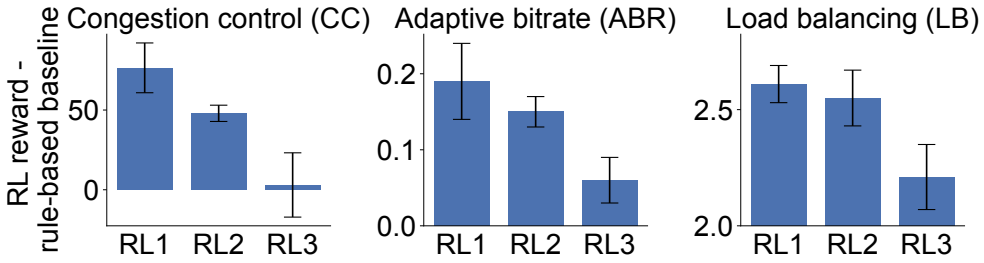
in networking. Revising the RL algorithm *per se* (input, output, or DNN model) is beyond our scope.

Network environments: We generate simulated training environments with a range of parameters, following prior work [101, 71, 102]. An environment can be synthetically generated using a list of parameters as *configuration*, e.g., in the context of ABR, a configuration encompasses bandwidth range, frequency of bandwidth change, chunk length, etc. Meanwhile, when recorded bandwidth traces are available (for CC and ABR experiments), we can also create *trace-driven* environments where the recorded bandwidth is replayed. Note that bandwidth is only one dimension of an environment and must be complemented with other synthetic parameters in order to create a simulated environment. (Our environment generator and a full list of parameters are documented in §A.2.) In recent papers, both trace-driven (e.g., [101, 58]) and synthetic environments (e.g., [71, 102]) are used to train RL-based network algorithms. We will explain in §2.4.2 how our technique applies to both types of environments.

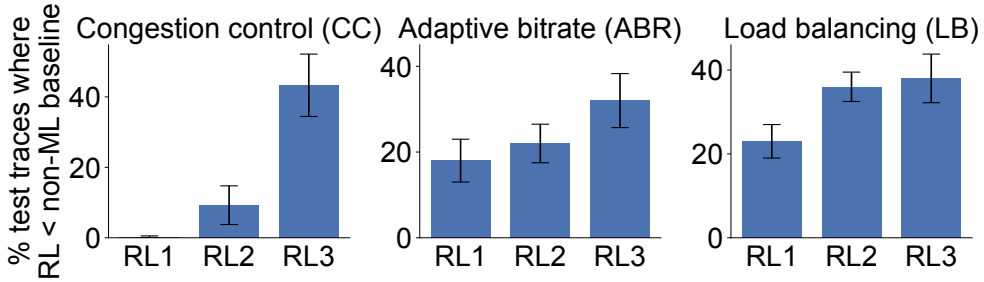
Traditional RL training: Given a user-specified distribution of (trace-driven or synthetic) training environments, the traditional RL training method works in iterations. Each iteration randomly samples a subset of environments from the provided distribution and then updates the DNN-based RL policy (via forward and backward passes). For instance, Aurora [71] uses an iteration of 7200 steps (i.e., 30–50 30-second network environments) and applies the PPO algorithm to update the policy network by simulating the network environments in each batch.

Several previous efforts have demonstrated the promise of the traditional RL training—given the distribution of target environments, an RL policy can be trained to perform well in these environments (e.g., [101, 71]). Unfortunately, this approach falls short on two fronts.

Challenge 1: Training over wide environment distributions.
When the training distribution of network environments has a widespread (e.g., a large range



(a) Performance gains of RL schemes over the baselines diminish as the target distribution spans a wide range of environments.



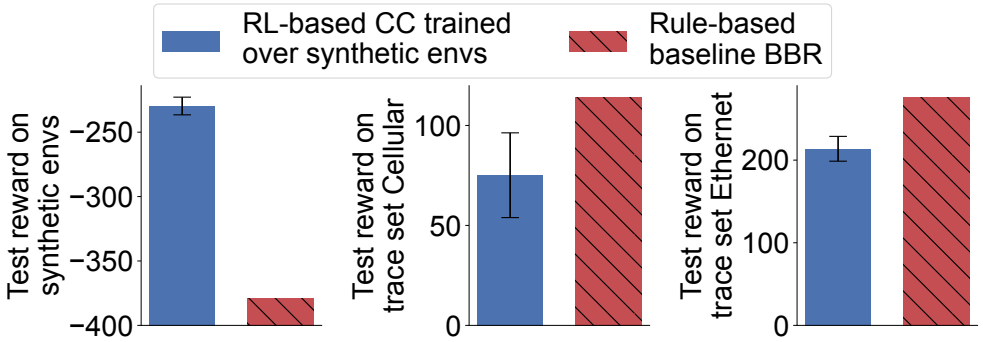
(b) Even if RL schemes perform better on average, they are worse than the baselines on a substantial fraction of test environments.

Figure 2.2: Challenges of RL training over a wider range of environments from small (RL1), medium (RL2), to large (RL3).

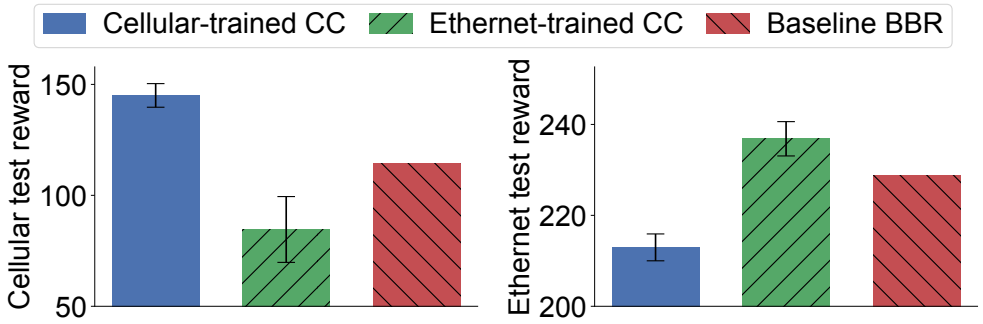
of possible bandwidth values), RL training tends to result in poor *asymptotic* performance (model performance after reaching convergence) even when the test environments are drawn from the *same* distribution as training.

In Figure 2.2, for each use case, we choose three target distributions (with increasing parameter ranges), labeled RL1/RL2/RL3 ranges of synthetic environment parameters in Table A.1, A.2, and A.3. Figure 2.2(a) compares the asymptotic performance of three RL policies (with different random seeds) with rule-based baselines, MPC [154] for ABR, BBR [33] for CC, and least-load-first (LLF) policy for LB, in test environments randomly sampled from the *same* ranges. It shows that RL's performance advantage over the baselines diminishes rapidly when the range of target environments expands. Even though RL-based policies still outperform the baselines on average, Figure 2.2(b) reveals a more striking reality—their performance falls behind the baselines in a substantial fraction of test environments.

An intuitive explanation is that in each RL training iteration, only a batch of randomly



(a) RL-based CC trained in synthetic network environments performs worse on real network traces than the rule-based baseline.



(b) RL-based CC trained over one real trace set performs worse on another real trace set than the rule-based baseline.

Figure 2.3: Generalization issues of RL-based schemes using CC as an example.

sampled environments (typically 20–50) is used to update the model, and when the entire training set spans a wide range of environments, the batches between two iterations may have dramatically different distributions which potentially push the RL model to different directions. This causes the training to converge slowly and makes it difficult to obtain a good policy [112]. Although this problem is not completely avoided in our solution, it is mitigated by curriculum learning which draws the environments of a batch from a "narrower" training environment distribution, thus reducing the discrepancies between batches.

Challenge 2: Low generalizability. Another practical challenge arises when the training process does not have access to the target environment distribution. This calls for models with good generalization, i.e., the RL policies trained on one distribution also perform well

on a different environment distribution during testing. Unfortunately, existing RL training methods often fall short of this ideal. Figure 2.3 evaluates the generalizability of RL-based CC schemes in two ways.

- First, we train an RL-based CC algorithm on the same range of synthetic environments as specified in its original paper [71]. We first validate the model by confirming its performance against a rule-based baseline BBR, in environments that are independently generated from the same range as training (Figure 2.3(a); left). Nevertheless, when tested on real-world recorded network traces under the category of "Cellular" and "Ethernet" from Pantheon [151] (Table 2.2), the RL-based policy yields much worse performance than the rule-based baseline.

- Second, we train the RL-based CC algorithm on the "Cellular" trace set and test it on the "Ethernet" trace set (Figure 2.3(b); left), or vice versa (Figure 2.3(b); right). Similarly, its performance degrades significantly when tested on a different trace set.

The observations in Figure 2.3 are not unique to CC. Prior work [58] also shows a lack of generalization of RL-based ABR algorithms.

Summary: In short, we observe two challenges faced by the traditional RL training mechanism:

- The asymptotic performance of the learned policies can be suboptimal, especially when they are trained over a wide range of environments.

- The trained RL policies may generalize poorly to unseen network environments.

2.3 Curriculum learning for networking

Given these observations regarding the limitations of RL training in networking, a natural question to ask is *how to improve RL training such that the learned adaptation policies*

achieve good asymptotic performance across a broad range of target network environments.[3]

Curriculum learning: We cast the training of RL-based network adaptation to the well-studied framework of curriculum learning. Unlike the traditional RL training that samples training environments from a *fixed* distribution in each iteration, curriculum learning *varies* the training environment distribution to gradually increase the difficulty of training environments, so that training will see more environments *that are more likely to improve*, which we refer to as *rewarding* environments. In many RL applications, prior work has shown the promise of curriculum learning, including faster convergence, higher asymptotic performance, and better generalization (§2.6).

The theoretical intuition behind curriculum learning is that a curriculum allows the model to optimize a family of gradually less smooth loss functions and prevents it from being trapped in local minima [26]. In the early stage of the curriculum, easier training samples are selected to comprise a smoothed loss function that reveals the big picture and is easier to optimize. The resulting model serves as a good starting point when more difficult samples are introduced to the training, reducing the smoothness of the loss function and making it harder to optimize. By optimizing the model on a sequence of loss functions with decreasing smoothness, the curriculum is able to gradually bring the model parameters close to the global optimum.

However, the challenge of employing curriculum learning lies in determining which environments are rewarding. Apparently, the answer to this question varies with applications, but three general approaches exist: (1) training the current model on a set of environments individually to determine in which environment the training progresses faster; (2) using heuristics to quantify the easiness of achieving model improvement an environment; and (3) jointly training another model (typically DNN) to select rewarding environments. Among

3. An alternative is to retrain the deployed RL policy whenever it meets a new domain (e.g., a new network connection with unseen characteristics), but this does not apply when the RL policy cannot be updated frequently. Besides, it is also challenging to precisely detect model drift in the network conditions that necessitate retraining the RL policy.

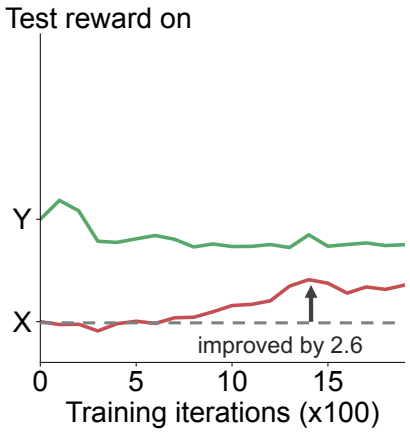
them, the first option is prohibitively expensive and thus not widely used, whereas the third introduces the extra complexity of training a second DNN. Therefore, we take a pragmatic stance and explore the second approach, while leaving the other two for future work.

Why sequencing training environments is difficult: A common strategy in curriculum learning for RL is to measure environment difficulty and gradually introduce more difficult environments to training. To motivate our design choices, we first introduce three strawman approaches, with different strengths and weaknesses. They are used to determine how rewarding an environment is. A good approach should always select network environments in which the RL model has a large improvement in reward when trained in them.

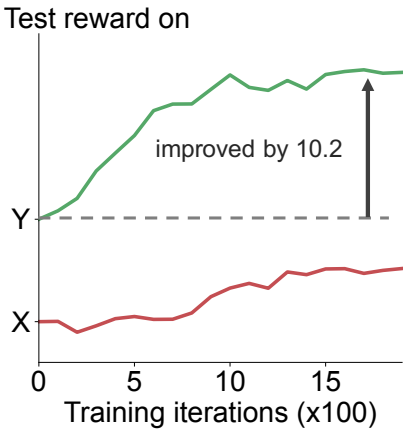
Strawman 1: inherent properties. The first idea is to quantify the difficulty level of an environment using some of its inherent properties. In congestion control, for instance, network traces with higher bandwidth variance are intuitively more difficult. This approach, however, only distinguishes environments that differ in the hand-picked properties and may not suffice under complex environments (e.g., adding bandwidth traces with similar variance to training can have different effects).

Strawman 2: performance of rule-based baselines. Alternatively, one can use the test performance of a traditional algorithm to indicate the difficulty of an environment. Lower performance may suggest a more difficult environment [146]. While this method can distinguish any two environments, it does not hint at how to improve the *current* RL model during training.

Strawman 3: performance gap to the optimum. To fix the problem of Strawman 2, one can use the performance gap between the current RL policy and the optimum instead [58]. If the current model performs much worse than the optimum in an environment (e.g., obtained by using ground-truth bandwidth as the bandwidth prediction), its performance might improve when trained in this environment. A caveat of this approach is that the computation of the optimal performance could be prohibitively expensive or even infeasible. This approach may

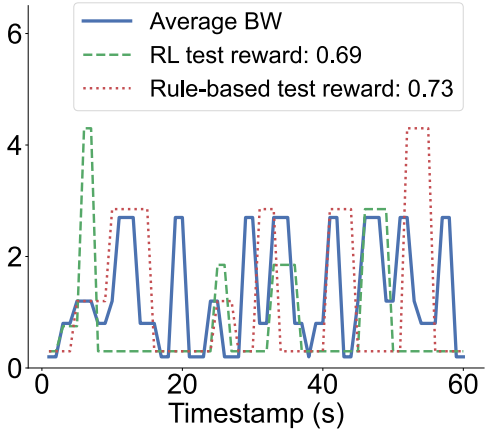


(a) Add X to training

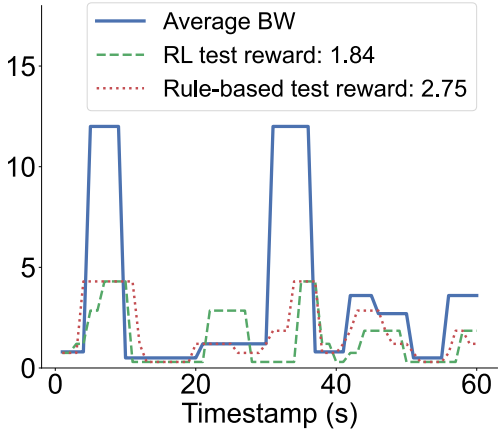


(b) Add Y to training

Figure 2.4: A simple example where adding trace set X to training has a different effect than adding Y. Adding X to training improves performance on X only marginally but hurts Y, whereas adding Y improves the performance on both X and Y.



(a) Trace in X (hard)



(b) Trace in Y (improvable)

Figure 2.5: Contrasting (a) an inherently hard (possibly unsolvable) environment with (b) an improvable environment. The difference is that the rule-based policy's reward is higher than the RL policy in (b), whereas their rewards are similar in (a).

also fail to improve RL's performance in environments that are inherently hard (e.g., highly fluctuating bandwidth in ABR and CC).

Example: Figure 2.4 shows a concrete example in ABR, where "Strawman 3" leads to a suboptimal outcome. (§2.5.5 will empirically test these three strawman approaches.) We first pretrain an RL-based ABR policy which performs poorly on X and Y (two sets of

bandwidth traces from two different environment configurations, details in §A.3). Since the performance gap between the current RL model and the optimum is larger on X than on Y, Strawman 3 opts for adding X to the training in the next step. However, Figure 2.4 shows that training further on X yields only a marginal reward improvement on X (and also hurts the performance on Y).

Instead, adding Y to training is a better choice at this point—the performance on Y is significantly improved (and it also benefits the performance on X though to a less extent).

To take a closer look, we plot two example traces from X and Y in Figure 2.5: The trace from X fluctuates with a smaller magnitude but more frequently, whereas the trace from Y fluctuates with a greater magnitude but much less frequently. However, such observations cannot generalize to an arbitrary pair of environments or a different application.

2.4 Design and implementation of GENET

2.4.1 Curriculum generation

To identify rewarding environments, the idea of GENET is to find environments with a large *gap-to-baseline*, i.e., the RL policy is worse than a given rule-based baseline by a large margin. At a high level, adding such environments to training has three practical benefits.

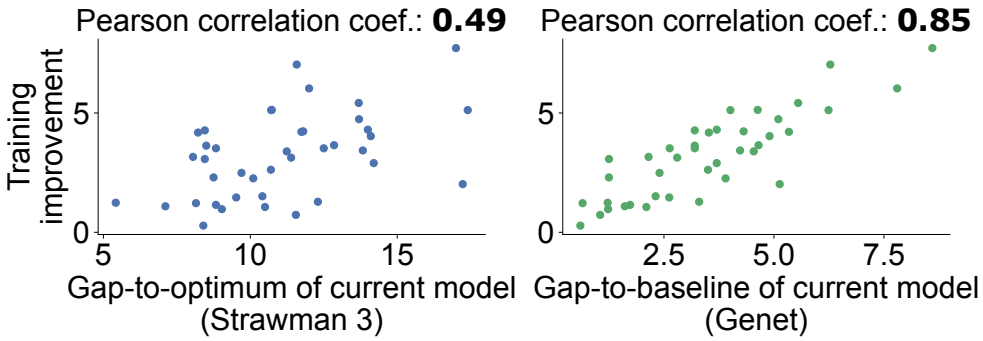
First, when a rule-based baseline performs much better than the RL policy in an environment, it means that the RL model may learn to "imitate" the baseline's known rules while training in the environment, bringing it on par with—if not better than—the baseline. [4] Therefore, a large gap-to-baseline indicates plausible room for the current RL model to improve. Figure 2.6 empirically confirms this with one example ABR policy and CC policy (both are intermediate models during GENET-based training). For example, among 73 randomly chosen synthetic environment configurations in CC, a configuration with a larger

4. This may not be true when the behavior of the rule-based algorithm cannot be approximated by RL's policy DNN, and we will discuss this issue in §2.7.

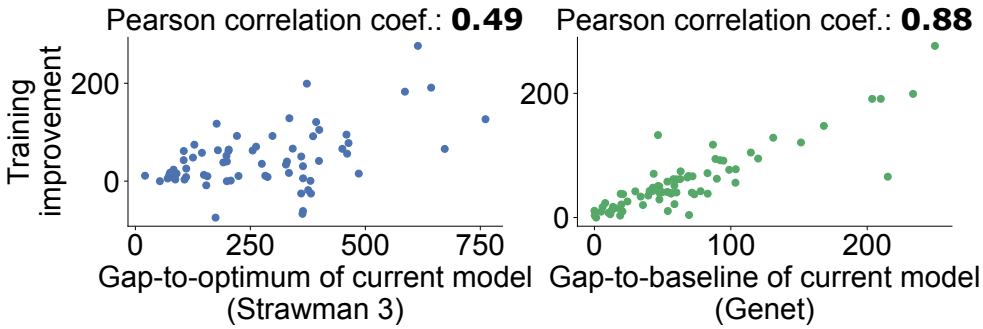
gap-to-baseline is likely to yield more improvement when adding its environments to the RL training. Moreover, this correlation is stronger than using the performance gap between the current model and the optimum ("Strawman 3" in §2.3) to decide which environments are rewarding. Nonetheless, the model's training improvement does not only depend on the gap-to-baseline. Other factors such as training hyperparameters can affect the reward improvement of an RL model. For example, too large a learning rate causes the RL model to jump over the optima while too small a learning rate slows down the convergence. In this work, we only focus on the gap-to-baseline and keep the training hyperparameters (e.g., learning rate, batch size of each iteration) unchanged in all the experiments.

Second, although not all rule-based algorithms are easily interpretable or completely fail-proof, many of them have traditionally been used in networked systems long before the RL-based approaches and are considered more trustworthy than black-box RL algorithms. Therefore, operators tend to scrutinize any performance disadvantages of the RL policy compared with the rule-based baselines currently deployed in the system. By promoting environments with large gap-to-baselines, GENET directly reduces the possibility that the RL policy causes performance regressions.

In short, the gap-to-baseline builds on the insight that rule-based baselines are *complementary* to RL policies—they are less susceptible to any discrepancies between training and test environments, whereas the performance of an RL policy is potentially sensitive to the environments seen during training. In §2.5.5, we will discuss the impact of different choices of rule-based baselines and why gap-to-baseline is a better way of using the rule-based baseline than alternatives. It is worth noting that the rewarding environments (those with large gap-to-baselines) do *not* have particular meanings outside the context of a given pair of RL model and baseline. For instance, when an RL-based CC model has a greater gap-to-baseline in some network environments, it only means that it is easier to improve the RL model by training it in these environments; it does not indicate if these environments are easy or



(a) ABR



(b) CC

Figure 2.6: Compared with the gap-to-optimum (left), the current model's gap-to-baseline (right) in an environment is more indicative of its potential training improvement in the environment.

challenging to any traditional CC algorithm.

2.4.2 Training framework

Figure 2.7 depicts GENET's high-level iterative workflow to realize curriculum learning. Each iteration consists of three steps (which will be detailed shortly):

1. First, we update the current RL model for a fixed number of iterations over the current training environment distribution;

2. Second, we select the environments where the current RL model has a large gap-to-baseline; and

3. Third, we promote these selected environments in the training environments distribu-

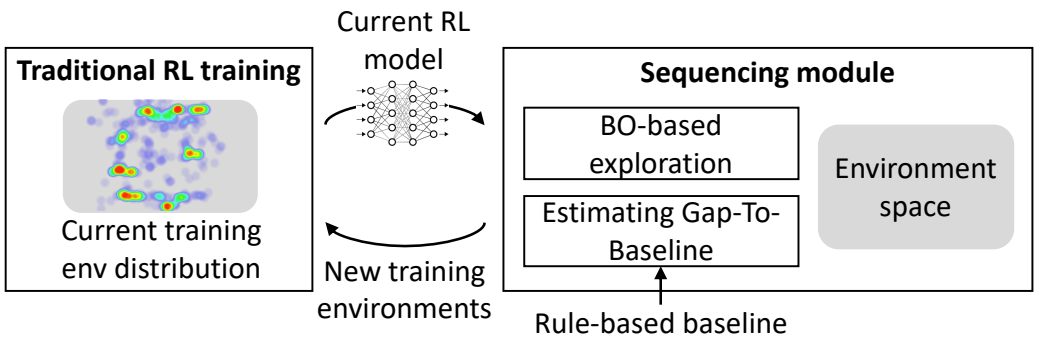


Figure 2.7: Overview of GENET's training process.

tion used by the RL training process in the next iteration.

Training environment distribution: We define a distribution of training environments as a probability distribution over the space of *configurations*, each being a vector of 5–6 parameters (summarized in Table A.1, A.2, A.3) used to generate network environments. An example configuration is: [BW: 2–3Mbps, BW changing frequency: 0–20s, Buffer length: 5–10s]. GENET sets the initial training environment distribution to be a uniform or exponential distribution along with each parameter, and automatically updates the distribution used in each iteration, effectively generating a training curriculum.

When recorded traces are available, GENET can augment the training with trace-driven environments as follows. Here we use bandwidth traces as an example. The first step is to categorize each bandwidth trace along with the bandwidth-related parameters (i.e., bandwidth range and variance in our case). Each time a configuration is selected by RL training to create new environments, with a probability of w (30% by default), GENET samples a bandwidth trace whose bandwidth-related parameters fall into the range of the selected configuration.

In §2.5.2, we will show that adding trace-driven environments to training improves the performance of RL policies, especially when tested in unseen real traces from the same distribution. That said, even if we do not use trace-driven environments in RL training, our trained RL policies still outperform the traditional method of training RL over real traces

or synthetic traces.

Key components: Each round of GENET starts with training the current model for a fixed number of iterations (defaults to 10). Here, GENET reuses the traditional training method in prior work (i.e., uniform sampling of training environments per iteration), which makes it possible to incrementally apply GENET to existing codebases (see our implementation in §2.4.3). Recent work on domain randomization [126, 138, 116] also shows that a similar training process can benefit the generalization of RL policies [126, 138, 116]. The details of the training process are described in Algorithm 1.

After a certain number of iterations, the current RL model and a pre-determined rule-based baseline are given to a *sequencing module* to search for the environments where the current RL model has a large gap-to-baseline. Ideally, we want to test the current RL model on all possible environments and identify the ones with the largest gap-to-baseline, but this is prohibitively expensive. Instead, we use *Bayesian Optimization* [56] (BO) as follows. We view the expected gap-to-baseline over the environments created by configuration p as a function of p: $Gap(p) = R(\pi^{rule}, p) - R(\pi^{rl}_{\theta}, p)$, where $R(\pi, p)$ is the average reward of a policy π (either the rule-based baseline π^{rule} or the RL model π^{rl}_{θ}) over k (10 by default) environments randomly generated by configuration p. BO then searches in the environment space for the configuration that maximizes $Gap(p)$.

Once a new configuration is selected, the environments generated by this configuration are then added to the training distribution as follows. When the RL training process samples a new training environment, it will choose the new configuration with w probability (30% by default) or uniformly sample a configuration from the old distribution with $1-w$ probability (70% by default), and then create an environment based on the selected configuration. Next, training is resumed over the new environment distribution.

It is important to notice that the BO-based search does *not* carry its states when searching rewarding environments for a new RL model. Instead, GENET restarts the BO search every

time the RL model is updated. The reason is that the rewarding environments can change once the RL model changes.

Design rationale: The process described above embeds several design decisions that make it efficient.

How to choose rule-based baselines? For GENET to be effective, the baselines should not fail in simple environments; otherwise, GENET would ignore them given that the RL policy could easily beat the baselines. For instance, when using Cubic as the baseline in training RL-based CC policies, we observe that the RL policy is rarely worse than Cubic along the dimension of random loss rate, because Cubic's performance is susceptible to random packet losses. That said, we find that the choice of baselines does not significantly impact the effectiveness of GENET, although a better choice tends to yield more improvement (as shown in §2.5.5).[5]

Why is BO-based exploration effective? GENET models the selection of network environments that maximize gap-to-baseline as a parameter search procedure in a high-dimensional space—each dimension of the space is a configuration of the network environment (e.g., link latency), each point in the space is a set of network environments with the same configurations, and the desired points are those whose environments have large gap-to-baselines. This problem has two features: (1) the environment search space is high-dimensional, and (2) evaluating the gap-to-baseline of a point in the space is computationally expensive (partly due to the variance among the environments with the same configurations). In this context, BO is merely one of the candidate solutions among several others to perform the parameter search. In §2.5.5, we will compare BO's efficiency with other candidate solutions and show that BO is efficient at identifying rewarding environments.

Why not set a threshold for the gap-to-baseline of the selected environments? While

5. One possible refinement in this regard is to use an "ensemble" of rule-based heuristics, and let the training scheduler focus on environments where the RL policy falls short of any one of a set of rule-based heuristics.

GENET uses BO to search rewarding environments with a fixed number of steps (default is 15), an alternative is to run BO until it finds an environment configuration whose gap-to-baseline is above a threshold. However, the latter strategy may not end (or take a long time to finish) if the RL model is already better than the baseline in most environments, which is possible during training. Moreover, the threshold introduces another hyperparameter to be tuned with domain knowledge.

Impact of forgetting? It is important that we train models over the *full* range of environments. GENET does begin the training over the whole space of environment in the first iteration, but each subsequent iteration introduces a new configuration, thus diluting the percentage of random environments in training. This might lead to the classic problem of forgetting—the trained model may forget how to handle environments seen before. While we do not address this problem directly, we have found that GENET is affected by this issue only mildly. The reason is that GENET stops the training after changing the training distribution for 9 times, and by then, the original environment distribution still accounts for about 10%.[6]

2.4.3 Implementation

GENET is fully implemented in Python and Bash, and has been integrated with three existing RL training codebases. Next, we describe the interface and implementation of GENET, as well as optimizations for eliminating GENET's performance bottlenecks.

API: GENET interacts with an existing RL training codebase with two APIs (Figure 2.8): `Train` signals the RL to continue the training using the given distribution of environment configurations and returns a snapshot of the model after a specified number of training iterations; `Test` calculates the average reward of a given algorithm (RL model or a baseline) over a specified number of environments drawn from the given distribution of configurations.

6. When we impose a minimum fraction of "exploration" (i.e., uniformly randomly picking an environment from the original training distribution) in the training (which is a typical strategy to prevent forgetting [156]), GENET's performance becomes worse.

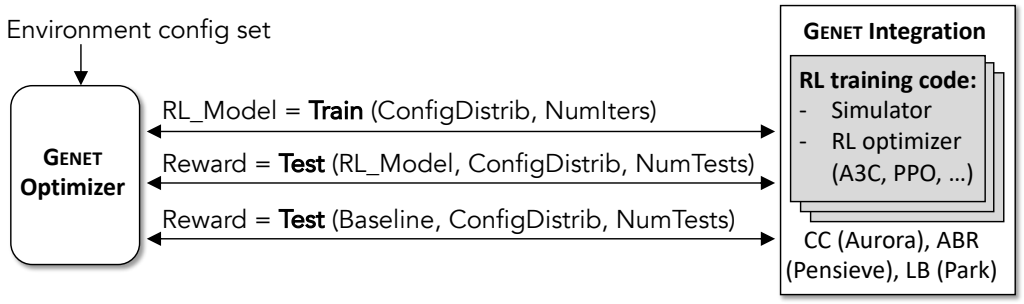


Figure 2.8: Components and interfaces needed to integrate GENET with an existing RL training codebase.

Name	Use case	Training # traces, total length (s)	Testing # traces, total length (s)
FCC	ABR	85, 105.8k	290, 89.9k
Norway	ABR	115, 30.5k	310, 96.1k
Ethernet	CC	64, 1.92k	112, 3.35k
Cellular	CC	136, 4.08k	121, 3.64k

Table 2.2: Network traces used in ABR and CC tests.

Integration with RL training: We have integrated GENET with Pensieve ABR [8], Aurora CC [2], and Park LB [7], which use different RL algorithms (e.g., A3C, PPO) and network simulators (e.g., packet level, chunk level). We implement the two APIs above using functionalities provided in the existing codebase.

Rule-based baselines: GENET takes advantage of the fact that many RL training codebases (including our three use cases) have already implemented at least one rule-based baseline (e.g., MPC in ABR, Cubic in CC) that runs in their simulators. In addition, we also implemented a few baselines by ourselves, including the shortest-job-first in LB, and BBR in CC. The implementation is generally straightforward, but sometimes the simulator (though sufficient for the RL policy) lacks crucial features for a faithful implementation of the rule-based logic. Fortunately, GENET-based RL training merely uses the baseline to select training environments, so the consequence of having a suboptimal baseline is not considerable.

2.5 Evaluation

The key takeaways of our evaluation are:

- Across three RL use cases in networking, GENET improves the performance of RL algorithms when tested in new environments drawn from the training distributions that include wide ranges of environments (§2.5.2).
- GENET improves the generalization of RL performance, allowing models trained over synthetic environments to perform well even in various trace-driven environments as well as on real-world network connections (§2.5.3).
- GENET-trained RL policies have a much higher chance of outperforming various rule-based baselines specified during GENET-based RL training (§2.5.4).
- Finally, the design choices of GENET, such as its curriculum learning strategy and BO-based search, are shown to be effective compared to seemingly natural alternatives (§2.5.5).

Given the success of curriculum learning in other RL domains, these improvements are not particularly surprising. However, by showing for the first time that curriculum learning facilitates RL training in networking, we hope to inspire more follow-up research in this direction.

2.5.1 Setup

We train GENET for three RL use cases in networking, using their original simulators: congestion control (CC) [2], adaptive bitrate streaming (ABR) [8], and load balancing (LB) [7]. As discussed in §2.4.1, we train and test RL policies over two types of environments.

Synthetic environments: We generate synthetic environments using the parameters described in detail in §A.2 and Table A.1,A.2,A.3. We choose these environment parameters

to cover a variety of factors that affect RL performance. For instance, in CC tests, our environment parameters specify bandwidth (e.g., the range, variance, and how often it changes), delay, queue length, etc.

Trace-driven environments: We also use real traces for CC and ABR (summarized in Table 2.2) to create trace-driven environments (in both training and testing), where the bandwidth time series are set by the real traces, but the remaining environment parameters (e.g., queue length or target video buffer length) are set as in the synthetic environments. We test ABR policies by streaming a pre-recorded video over 290 traces from FCC broadband measurements [45] (labeled "FCC") and 310 cellular traces [122] (labeled "Norway"). We test CC policies on 121 cellular traces (labeled "Cellular") and 112 Ethernet traces (labeled "Ethernet") collected by the Pantheon platform [151].

Baselines: We compare GENET-trained policies with several baselines. First, *traditional RL* trains RL policies by uniformly sampling environments from the target distribution per iteration. We train three types of RL policies (RL1, RL2, RL3) over fixed-width uniform distribution of synthetic environments, specified in Table A.1, A.2, A.3. From RL1 to RL3, the sizes of their training environment ranges are in ascending order.

We also train RL policies over trace-driven environments, i.e., randomly picking bandwidth traces from one of the recorded sets. This is the same as prior work, except that we also vary non-bandwidth-related parameters (e.g., queue length, buffer length, video length, etc) to increase its robustness. In addition, we test an early attempt to improve RL [58] which generates new training bandwidth traces that maximize the gap between the RL policy and optimal adaptation with a non-smoothness penalty (§2.5.5).

Second, *traditional rule-based algorithms* include BBA [67] and RobustMPC [154] for ABR, PCC-Vivace [49], BBR [33] and CUBIC for CC, and least-load-first (LLF) for LB.[7]

7. By default, we use RobustMPC as MPC and PCC Vivace-latency as Vivace, since they appear to perform better than their perspective variants.

They can be viewed as a reference point for traditional non-ML solutions.

2.5.2 Asymptotic performance

We first compare GENET-trained policies and traditionally trained RL policies, in terms of their *asymptotic performance* (i.e., test performance over new test environments drawn independently from the training distribution). In other words, we train RL policies over environments from the target distribution and test them in new environments from the same distribution.

Synthetic environments: We first test GENET-trained CC, ABR, and LB policies under their perspective RL3 synthetic ranges (where all parameters are set to their full ranges) as the target distribution. As shown in Figure 2.2, in these training ranges, traditional RL training yields little performance improvement over the rule-based baselines. Figure 2.9 compares GENET-trained CC, ABR, and LB policies with their respective baselines over 200 new synthetic environments randomly drawn with the target distribution.

Across three use cases, we can see that GENET consistently improves over traditional RL-trained policies by 8–25% for ABR, 14–24% for CC, 15% for LB, compared with traditional RL training methods. We notice that there is no clear ranking among the three traditional RL-trained policies. This is because RL1 helps training to converge better but only sees a small slice of the target distribution, whereas RL3 sees the whole distribution but cannot train a good model. In contrast, GENET outperforms them, as curriculum learning allows it to learn more efficiently from the large target distribution.

To show the performance more thoroughly, Figure 2.10 picks ABR as an example and shows the performance across different values along with six environment parameters. We vary one parameter at a time while fixing other parameters at the same default values (see Table A.1, A.2, A.3). We see that GENET-trained RL policies enjoy consistent performance advantages (in reward) over the RL policies trained by traditional RL-trained models. This

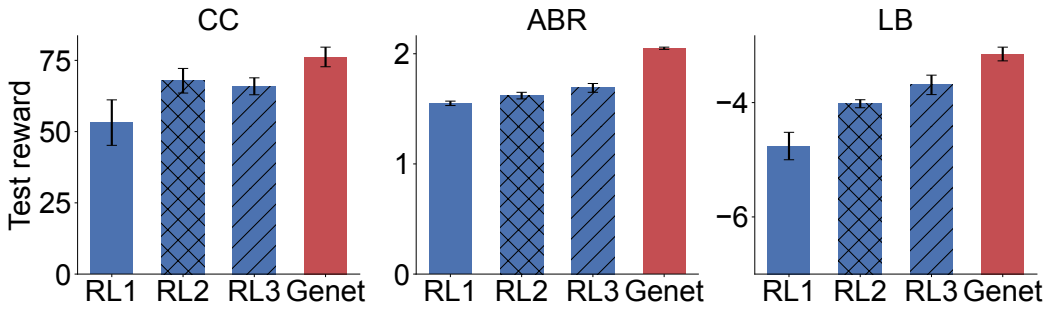


Figure 2.9: Comparing the performance of GENET-trained RL policies for CC, ABR, and LB, with baselines in unseen synthetic environments drawn from the training distribution, which sets all environment parameters to their full ranges.

suggests that the improvement of GENET shown in Figure 2.9 is not a result of improving rewards in some environments at the cost of degrading rewards in others; instead, GENET improves rewards in most cases. Figure 2.11 shows that in the simulated environments [7], the GENET-trained LB policy outperforms its baselines by 15%.

Trace-driven environments: Next, we set the target environment distributions of ABR and CC to be the environments generated from multiple real-world trace sets (FCC and Norway for ABR, Ethernet and Cellular for CC). We partition each trace set as listed in Table 2.2. GENET trains ABR and CC policies by combining trace-driven environments and synthetic environments (described in §2.4.2). For a thorough comparison, both GENET and the traditional RL training have access to the training portion of the real traces as well as the synthetic environments. We vary the ratio of real traces and synthetic environments and feed them to the traditional RL training method, e.g., if the ratio of real traces is 20%, then the traditional RL training randomly draws a trace-driven environment with 20% probability and synthetic environments with 80% probability. That is, we test different ways for the traditional RL training to combine the training traces and synthetic environments. Figure 2.12 tests GENET-trained ABR and CC policies with their respective traditional RL-trained baselines over new environments generated from the traces in the testing set. Figure 2.12 shows that GENET-trained policies outperform traditional RL training by 17–

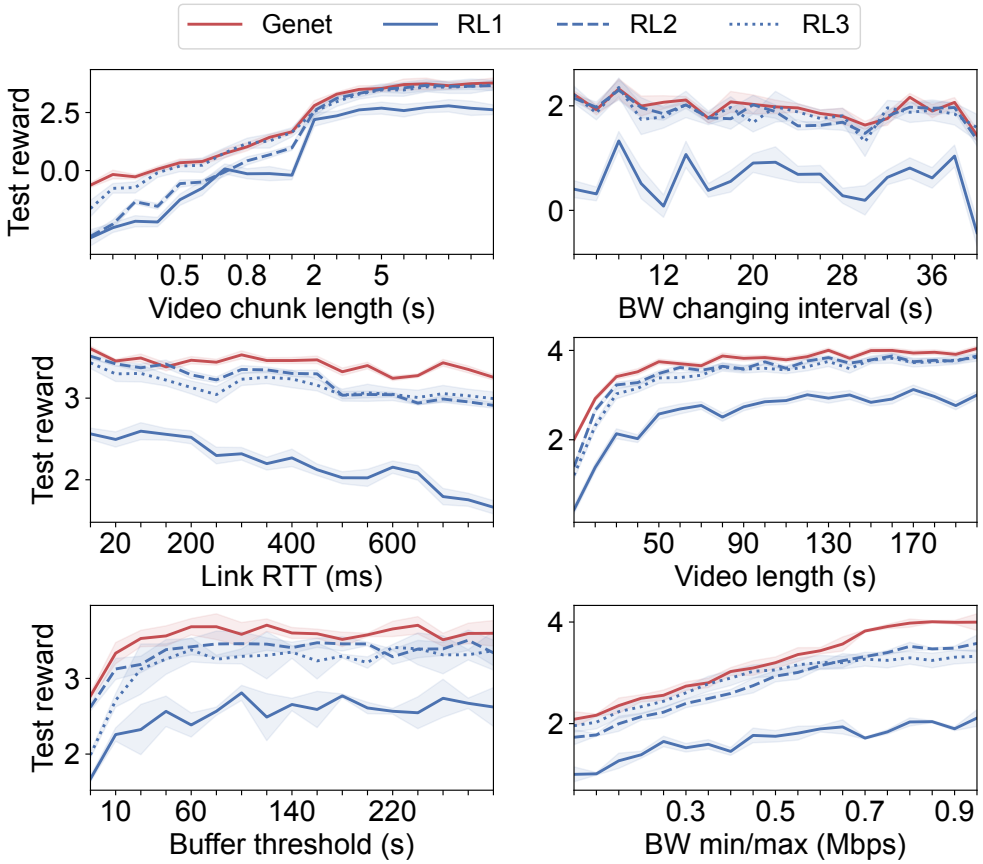


Figure 2.10: Test of ABR policies along individual env-parameters.

18%, regardless of the ratio of real traces, including when training the model entirely on real traces.

2.5.3 Generalization

Next, we take the RL policies of ABR and CC trained (by GENET and other baselines) entirely over synthetic environments (the RL3 synthetic environment range) and test their generalization in trace-driven environments generated by the ABR (and CC) testing traces in Table 2.2.

Figure 2.13 shows that they perform better than traditional RL baselines trained over the same synthetic environment distribution. Though Figure 2.13 uses the same testing environments as Figure 2.12 and has a similar relative ranking between GENET and traditional

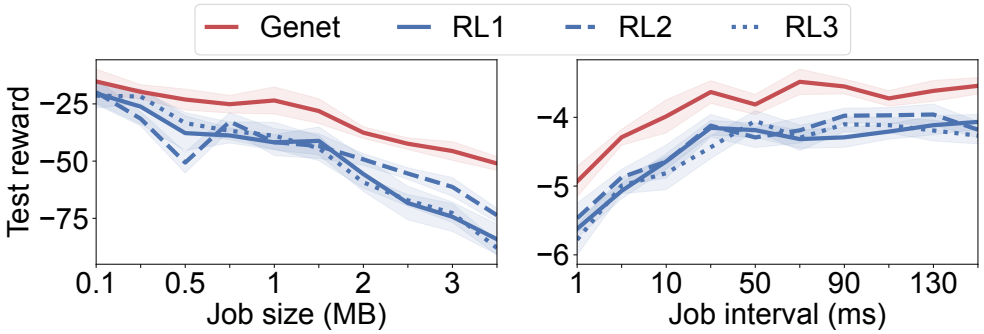
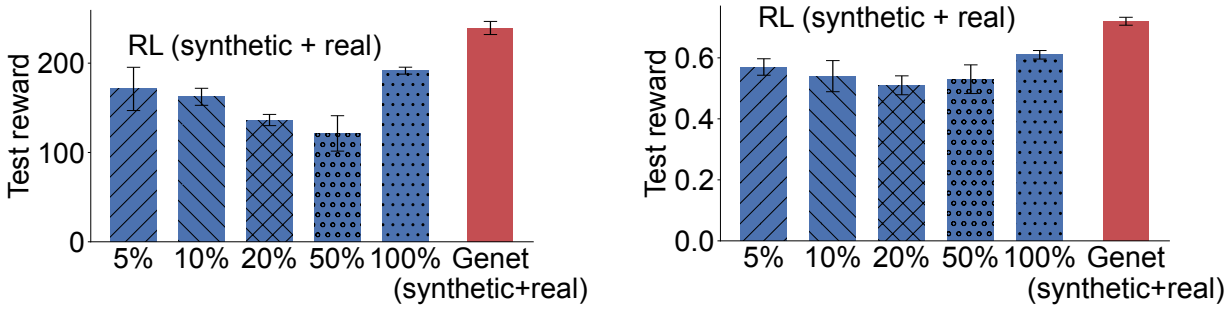


Figure 2.11: Test of LB policies along individual env-parameters.



(a) Congestion control (CC)

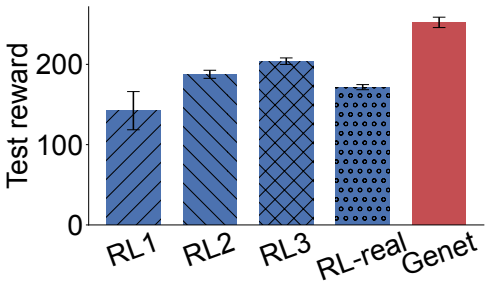
(b) Adaptive bitrate (ABR)

Figure 2.12: Asymptotic performance of GENET-trained CC policies (a) and ABR policies (b) and baselines, when the real network traces are randomly split into a training set and a test set.

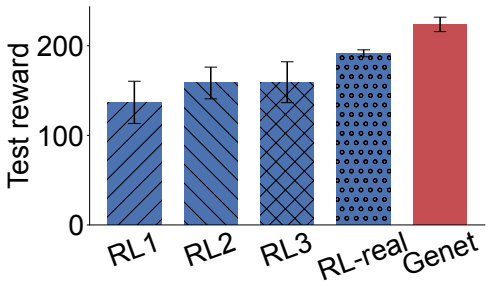
RL training, the implications are different: Figure 2.13 also shows that when the real traces are *not* accessible in training, GENET can produce models with better generalization in real-trace-driven environments than the baselines, whereas Figure 2.12 shows their performance when the real traces are actively used in training of GENET and the baselines.

2.5.4 Comparison with rule-based baselines

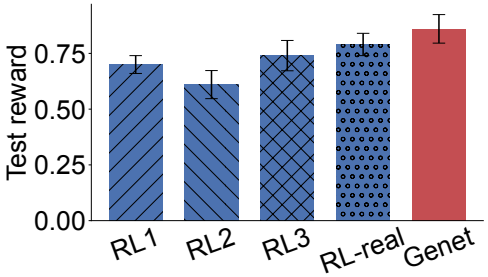
Impact of the choice of rule-based baselines: Figure 2.14 shows the performance of GENET-trained policies when using different rule-based baselines. We choose MPC and BBA as baselines in the ABR experiments and BBR and Cubic as baselines in CC experiments, respectively. We observe that in all cases, GENET-trained policies outperform their respective rule-based baselines.



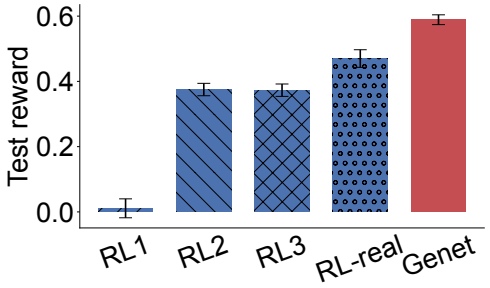
(a) CC test in trace-driven environments (Cellular)



(b) CC test in trace-driven environments (Ethernet)



(c) ABR test in trace-driven environments (FCC)



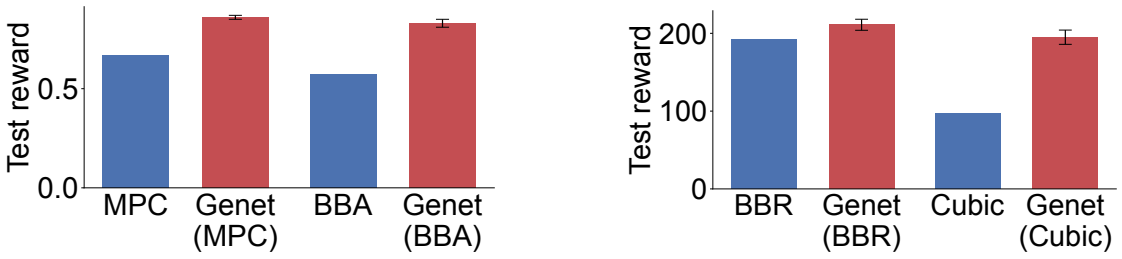
(d) ABR test in trace-driven environments (Norway)

Figure 2.13: Generalization test: Training of various methods is done entirely in synthetic environments, but the testing is over various real network trace sets.

What if GENET uses naive rule-based baselines? As explained in §2.4.2, the rule-based baseline should have a reasonable (though not necessarily optimal) performance; otherwise, it would be unable to indicate when the RL policy can be improved. To empirically verify it, we use two unreasonable baselines: choosing the highest bitrate when rebuffer in ABR, and choosing the highest loaded server in LB. In both cases, the BO-based search fails to find useful training environments, because the RL policy very quickly outperforms the naive baseline everywhere. That said, the negative impact of using a naive baseline is restricted to the selection of training environments, rather than the RL training itself (a benefit of decoupling baseline-driven environment selection and RL training), so in the worst case, GENET would be roughly as good as traditional RL training.

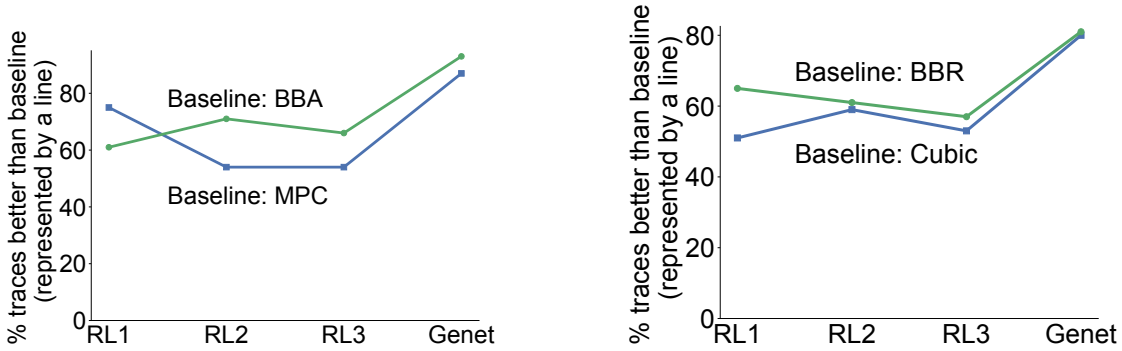
How likely is GENET to outperform rule-based baselines?

One of GENET's benefits is to increase how often the RL policy is better than the rule-based



(a) ABR (b) CC

Figure 2.14: *GENET outperforms the rule-based baselines used in its training.*

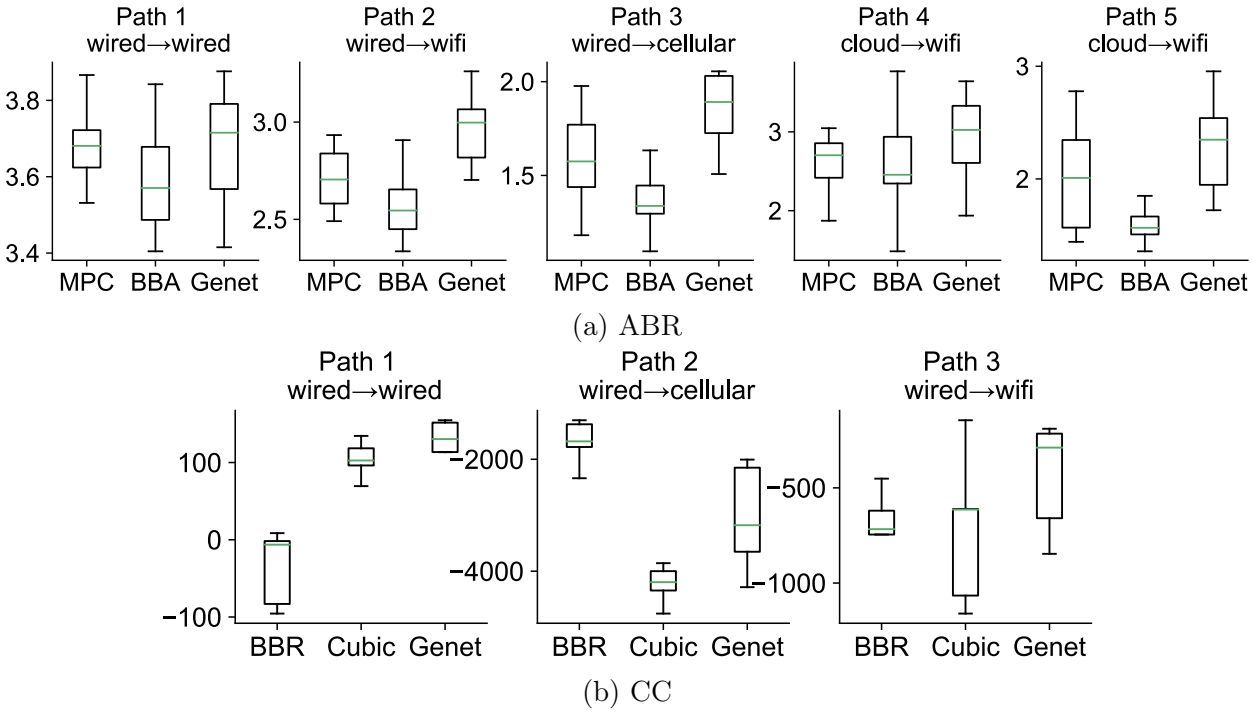


(a) ABR (b) CC

Figure 2.15: Fraction of real traces where GENET-trained policies (and traditional RL) are better than the rule-based baselines.

baseline used in GENET. In Figure 2.15, we create various versions of GENET-trained RL policies by setting the rule-based baselines to be Cubic and BBR (for CC), and MPC and BBA (for ABR). Compared to RL1, RL2, RL3 (unaware of rule-based baselines), GENET-trained policies remarkably increase the fraction of real-world traces (emulated) where the RL policy outperforms the baseline used to train them. This suggests that operators can specify a rule-based baseline, and GENET will train an RL policy that outperforms it with high probability.

Breakdown of performance: Figure 2.17 takes one GENET-trained ABR policy (with MPC as the rule-based baseline) and one GENET-trained CC policy (with BBR as the rule-based baseline) and compares their performance with a range of rule-based baselines along with individual performance metrics. We see that the GENET-trained ABR and CC policies



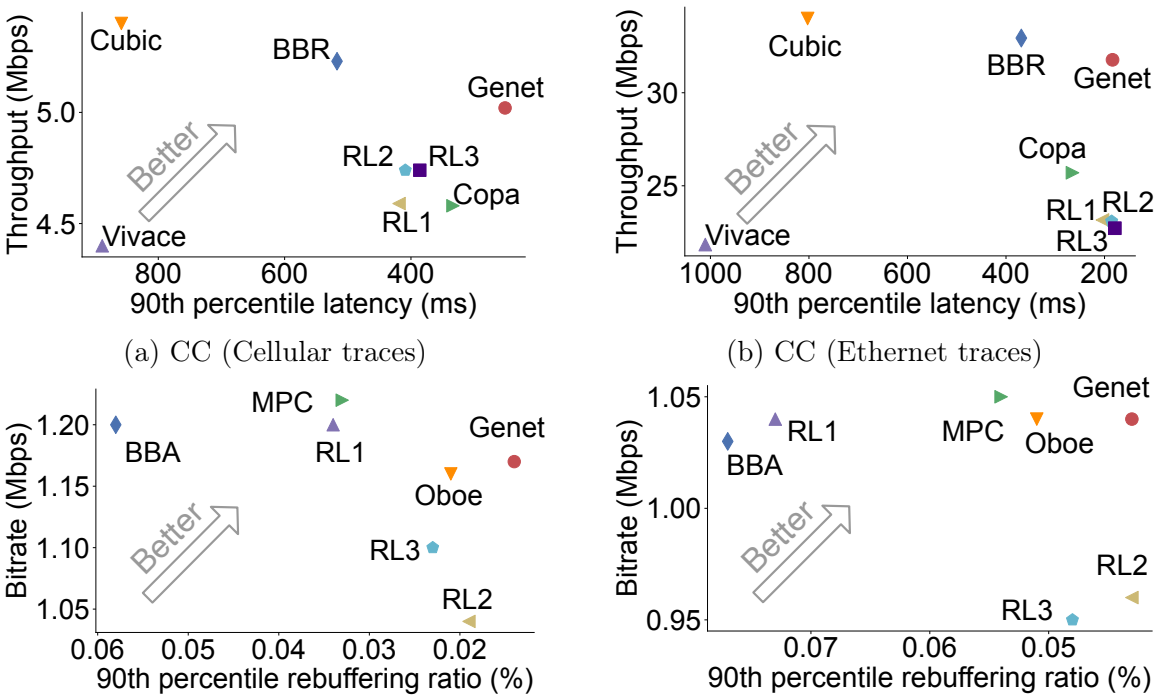
(a) ABR

(b) CC

Figure 2.16: Testing ABR and CC policies in real-world environments.

stay on the frontier and outperform other baselines.

Real-world tests: We also test the GENET-trained ABR and CC policies in five real wide-area network paths (without emulated delay/loss), between four nodes reserved from OpenNetLab [6, 53], one laptop at home, and two cloud servers (§A.4), allowing us to observe their interactions with real network traffic. For statistical confidence, we run the GENET-trained policies and their baselines back-to-back, each at least five times, and show their performance in Figure 2.16. The system metrics behind each reward value are shown in Table A.4 and Table A.5. In all but two cases, GENET outperforms the baselines. On Path-2, GENET-trained ABR has little improvement, because the bandwidth is always much higher than the highest bitrate, and the baselines will simply use the highest bitrate, leaving no room for improvement. On Path-3, GENET-trained CC has negative improvement, because the network has a deeper queue than used in training, so RL cannot handle it well. This is an example where GENET can fail when tested out of the range of training environments.



(a) CC (Cellular traces)

(b) CC (Ethernet traces)

(c) ABR (FCC traces)

(d) ABR (Norway traces)

Figure 2.17: RL-based ABR and CC vs. rule-based baselines.

These results do not prove that the policies generalize to all environments; instead, they show GENET's performance in a range of network settings.

2.5.5 Understanding GENET's design choices

Alternative curriculum-learning schemes: Figure 2.18 compares GENET's training curve with that of traditional RL training and three alternatives for selecting training environments described in §2.3. **CL1** uses hand-picked heuristics (gradually increasing the bandwidth fluctuation frequency in the training environments), **CL2** uses the performance of a rule-based baseline (gradually adding environments where BBR for CC and MPC for ABR performs badly), and **CL3** adds traces where the current RL model is much worse than the optimum (whereas GENET picks the traces where the current RL model is much worse than a rule-based baseline). Compared to these baselines, In Figure 2.18, we show

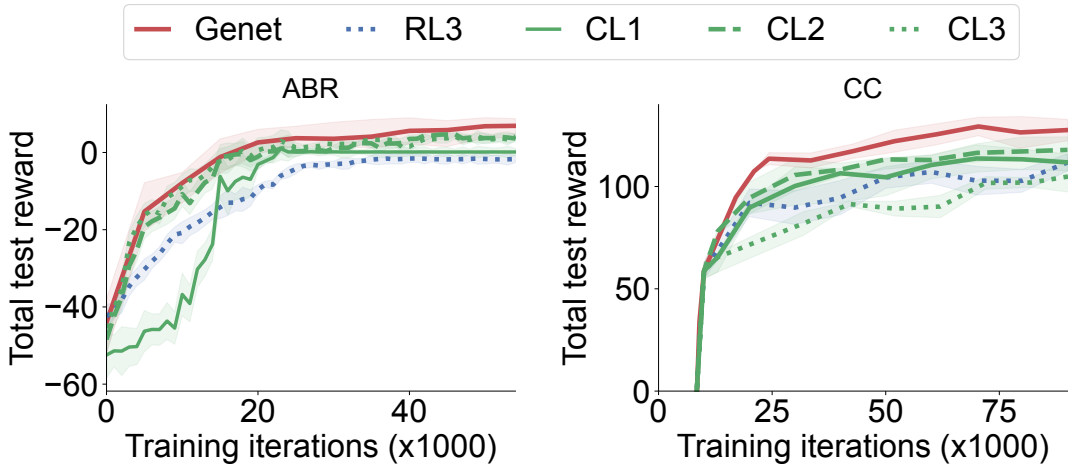


Figure 2.18: GENET's training ramps up faster than alternative curriculum learning strategies.

that GENET's training curves have faster ramp-ups, suggesting that with the same number of training iterations, GENET can arrive at a much better policy, which corroborates the reasoning in §2.3.

In addition, "Robustifying" [58][8] (which learns an adversarial bandwidth generator) also tries to improve ABR logic by adding more challenging environments to training. For a more direct comparison with GENET, we implement a variant of GENET where BO picks configurations that maximize the gap between RL and the optimal reward (penalized by bandwidth non-smoothness with different weights of p). Figure 2.19 compares the resulting RL policies with GENET-trained RL policy and MPC as a baseline on the synthetic traces in Figure 2.10. We see that they perform worse than GENET-trained ones and that by changing the BO's environment selection criteria, GENET becomes less effective. GENET outperforms Robustifying, because the non-smoothness metric used in [58] may not completely capture the inherent difficulty of bandwidth traces (Figure 2.5 shows a concrete example).

BO-based search efficiency: GENET uses BO to explore the multi-dimensional environ-

8. In lack of a public implementation, we follow the description in [58] (e.g., non-smoothness weight) and apply it to Pensieve (with the only difference being that for fair comparisons with other baselines, we apply it on Pensieve trained on our synthetic training environments). We have verified that our implementation of Robustifying achieves similar improvements in the setting of original paper. More details are in Appendix A.6.

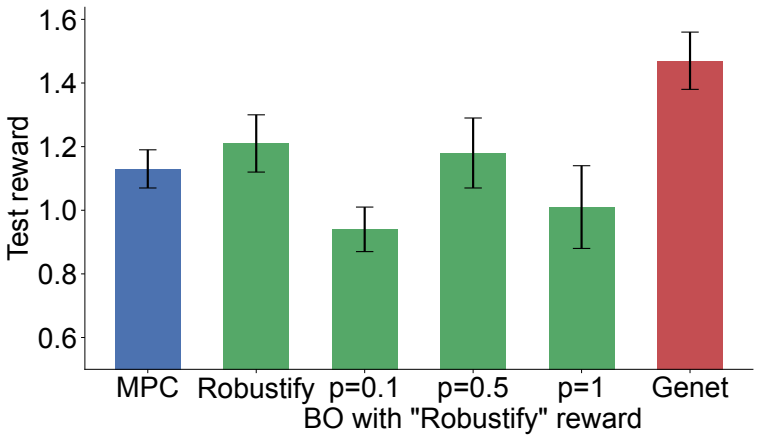
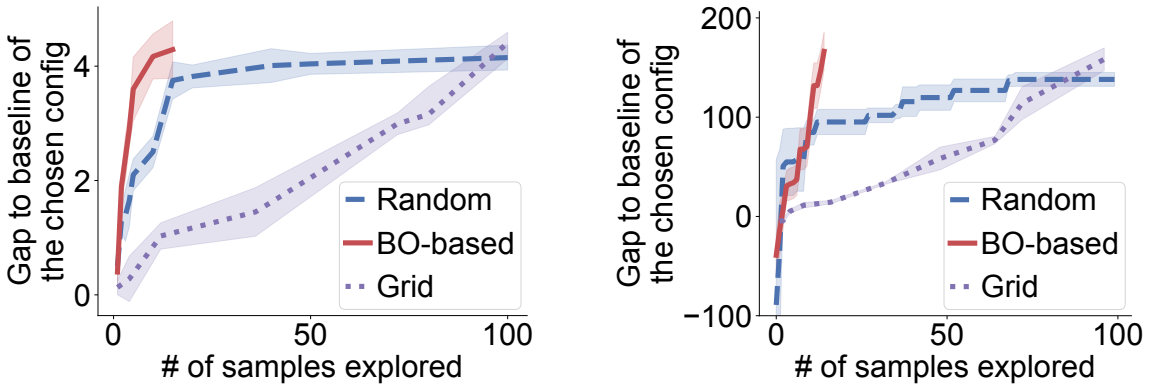


Figure 2.19: GENET outperforms Robustifying [58] that improves RL performance by generating adversarial bandwidth traces, and variants of GENET using Robustifying's criteria in BO-based environment selection.

ment space environment to find the environment configuration with a large gap-to-baseline. While BO may not always find the single optimal point in arbitrary blackbox function between environment parameters and gap-to-baseline, we found it to be a pragmatic solution. To show it, we randomly choose an intermediate RL model during the GENET training of ABR and CC. Figure 2.20 shows the gap-to-baseline of the configuration selected by BO for each model within 15 search steps. Within a small number of steps, it can identify a configuration that is almost as good as randomly searching for 100 points, which is much more expensive. Figure 2.20 also includes the grid search as a reference, which starts with all configurations initialized to their respective midpoints and then searches and updates the best value for each configuration one by one. We observe that it does not converge as fast as BO.

2.6 Related work

Improving RL for networking: Some of our findings regarding the lack of generalization corroborate those in previous work [148, 101, 71, 58, 124, 48]. To improve RL for networking use cases, prior work has attempted to apply and customize techniques from the ML



(a) ABR (b) CC

Figure 2.20: *BO-based search is more efficient at finding environments with large gap-to-baselines than random exploration in the environment configuration space.*

literature. For instance, [58] applies adversarial learning by generating relatively smooth bandwidth traces that maximize the RL regret w.r.t. optimal outcomes, [52, 82] show that the generalization of RL can be improved by incorporating training environments where a given RL policy violates pre-defined safety conditions, [129, 128] incorporate randomization in the evaluation of RL-based systems, and Fugu [153] achieves a similar goal through learning a transmission time predictor *in situ*. Other proposals seek to safely deploy a given RL policy in new environments [103, 124, 132]. In many ways, GENET follows this line of work, but it is different in that it systematically introduces curriculum learning, which has underpinned many recent enhancements of RL and demonstrates its benefits across multiple applications.

Curriculum learning for RL: There is a substantial literature on improving deep RL with curricula ([112, 61, 118] give more comprehensive surveys on this subject). Each component of curriculum learning has been extensively studied, including how to generate tasks (environments) with potentially various difficulties [133, 130], how to sequence tasks [121, 135], and how to add a new task to training (transfer learning). In this work, we focus on sequencing tasks to facilitate RL training. It is noticed that, for general tasks that do not have a clear definition of difficulty (like networking tasks), optimal task sequencing is still an

open question. Some approaches, such as self-paced learning [85] advocate the use of easier training examples first, while the other approaches prefer to use harder examples first [37]. Recent work tries to bridge the gap by suggesting that an ideal next training task should be difficult for the current model's hypothesis, while it is also beneficial to prefer easier points with respect to the target hypothesis [61]. In other words, we should prefer an easy environment that the current RL model cannot handle well, which confirms the intuition elaborated in Bengio's seminal paper [26], which hypothesizes that "it would be beneficial to make learning focus on 'interesting' examples that are neither too hard nor too easy." GENET is an instantiation of this idea in the context of networking adaptation, and the way to identify the rewarding (or "interesting") environments is by using the domain-specific rule-based schemes to identify where the current RL policy has a large room for improvement.

Automatic generation of curricula also benefits generalization, particularly when used together with domain randomization [116]. Several schemes boost RL's training efficiency by iteratively creating a curriculum of challenging training environments (e.g., [47, 106]) where the RL performance is much worse than the optimal outcome (i.e., maximal regret). When the optimal policy is unavailable, they learn a competitive baseline [47] to approximate the optimal policy or a metric [106] to approximate the regret. GENET falls in this category, but proposes a domain-specific way of identifying rewarding environments using rule-based algorithms.

Some proposals in safe policy improvement (SPI) for RL also use rule-based schemes [57, 88], though for different purposes than GENET. While GENET uses the performance of rule-based schemes to identify where the RL policy can be maximally improved, SPI uses the decisions of rule-based algorithms to avoid violation of failures during training.

2.7 Discussion

Does a small gap-to-baseline always mean that an RL model has small improvement when trained on it?

Although a small gap-to-baseline on a network environment indicates that the RL model already performs quite closely with the rule-based baseline, there is still a chance that the RL model could be greatly improved when trained in that environment. This is because if the rule-based baseline performs very badly in an environment, the gap-to-baseline will no longer be indicative of the potential improvement of RL training. For example, Cubic may perform poorly on a high-bandwidth link with occasional random packet loss, as Cubic does not differentiate random packet loss and congestion-induced loss, causing it to lower congestion window size when the available bandwidth does not drop. In such cases, even if an RL model has a small gap-to-baseline with Cubic, there *could* still be room for the RL model to improve performance, but GENET may not choose to prioritize such environments. That said, this problem could be mitigated by using a more performant baseline or an "ensemble" of existing baselines (i.e., measuring the maximum gap to *any* baseline from a set).

Does training in environments of large gap-to-baseline always lead to large RL model improvement?

Unfortunately, the answer is not always. RL models may not always be able to approximate the performance of rule-based baselines, e.g., due to an RL model's coarse decision granularity. For instance, Aurora (an RL-based CC) is a monitor-interval-based CC algorithm. Each monitor interval needs to be long enough to accumulate enough packet acks (e.g., 10–50) to compute the features (throughput, latency, etc.) for the RL model to select the sending rate. In contrast, traditional TCP algorithms like Cubic and BBR can update sending rate (cwnd) on the arrival of each packet ack. Thus, Aurora has a much coarser decision granularity than traditional TCPs, rendering it hard for the RL model to approximate the traditional TCP's behavior when the network condition suddenly changes. For instance, during sudden band-

width drops and rapid queue buildups, the inter-packet interval dramatically increases, and so does Aurora's monitor interval, whereas TCP Cubic or BBR can still update its sending rate on each packet ack. In these cases, Aurora will never ramp up or reduce sending rate as fast as its rule-based baselines, so even with a large gap-to-baseline in such environments, Aurora may not see a large reward improvement.

What if a rule-based baseline does not exist?

The current GENET training framework requires the existence of a rule-based baseline for the target networking problem. If the problem does not have a well-studied rule-based baseline, there are three alternative training methods that GENET can fall back to. First, GENET can fall back on traditional RL training. Although it loses the benefits of curriculum learning, it may still produce a reasonable RL-based policy. Second, we can use the performance gap between an optimal solution based on ground truth knowledge (such as future bandwidth variation) and the current RL model as the guidance of rewarding network environment selection. [58] trains an ABR RL model using network traces from a bandwidth-generating model. The training of the bandwidth-generating model is then guided by the performance gap between the optimal solution and the current RL model. This training method works well when the optimal solution is feasible and computationally cheap. Third, a trained RL model can be treated as a rule-based baseline. [47] trains two RL models (with identical model architecture) competitively on the environments produced by an adversarial generator. The adversarial generator is a neural network that aims to maximize the reward difference between the two RL models. However, the training complexity increases due to the increased number of models to be trained. Even though GENET can fall back on alternative training methods, how to extend it to work in applications domains that do not have an existing rule-based baseline remains to be investigated.

2.8 Conclusion

We present GENET, a new training framework to improve the training of deep RL-based network adaptation algorithms. For the first time, we introduce curriculum learning to the networking domain as the key to reaching better RL performance and generalization. To make curriculum learning efficient in networking, the main challenge is how to automatically identify the "rewarding" environments that can maximally benefit from retraining. GENET addresses this challenge with a simple-yet-efficient idea that highly rewarding network environments are where the current RL performance falls significantly behind that of a rule-based baseline scheme. Our evaluation on three RL use cases shows that GENET improves RL policies (in both performance and generalization) in various environments and workloads.

CHAPTER 3

LOSS-TOLERANT NEURAL VIDEO CODEC AWARE CONGESTION CONTROL FOR REAL TIME VIDEO COMMUNICATION

3.1 Introduction

Real-time video communication including video conferencing [97], live video/VR broadcasting [16, 12, 64], IoT applications [1, 17], and cloud gaming [15, 13] has been a key component of our daily lives [29] and carries a dominant amount of traffic in today's internet [42].

These RTC applications require high network bandwidth and low network latency to deliver seamless and high quality experience to users, pushing the telecommunication infrastructure upgrade to meet the demands and forcing the congestion control (CC) algorithms to promptly adapt to the constantly changing network conditions. Unlike traditional congestion controls [60, 31, 69] which are designed for reliability and in-order delivery through retransmissions instead of realtimeliness, plenty of hand-crafted congestion controls for real-time video communication [34, 55, 162, 78, 110, 120] have been proposed to boost bandwidth utilization, suppress packet delays, and avoid packet losses. However, these pre-programmed rule-based congestion control algorithms are not panacea in all network settings as they fall short of adapting to the highly heterogeneous network conditions.

To save the human effort optimizing a rule-based congestion control algorithm for numerous network conditions, researchers have made huge effort to explore the data-driven approaches to design congestion control and rate adaptation [101, 71, 157, 158, 159, 150, 58], which have shown great potential over the handcrafted heuristics. The "learning online, running online" strategy is often adopted to train a RL-based solution in order to bridge the gap between training network environments and the real network environments at the deployment stage. RL models directly interact with the real network environments to col-

lect experience and then update themselves during runtime. However, the trial-and-error behavior of online RL training will unavoidably take risky actions which might disturb the system performance. A safeguard policy, typically a handcrafted heuristics, is often used to substitute the RL model once an erroneous action is detected or the system is in a risky state [157, 103]. After the safeguard policy recovers the system to a safe state, the RL-based model takes the control back.

The main design philosophy behind safeguarding an RL-based CC by traditional CCs in RTC applications is based on an implicit assumption on video codec. The assumption is that delayed or lost packets can lead to incomplete frames received which then block video decoding at the receiver side and hurt users' QoE badly. Nevertheless, the existence of safeguard policy might slow down the RL model action space exploration and hinder the learning progress.

The recent loss-tolerant neural video codecs (NVC) [46, 65, 94, 39, 38, 134] breaks the implicit assumption on video codecs as these NVCs can decode incomplete frames and still deliver decent frame quality. They have shown strong loss tolerance ability across a wide range of packet loss rates on top of its high compression efficiency and good generalization over various video content. Figure 3.1 shows a state-of-art NVC, GRACE, has a smoother and slower video quality drop with increasing packet loss rate than commonly used encoder-side forward error correction (FEC) and decoder-side error concealment (EC).

However, there is no systematic way to adapt the congestion control algorithm to best utilize the loss-tolerant video codecs. The traditional RL-based congestion controls fall short of being aware of video codec properties as their RL rewards are typically a linear combination of network throughput, packet delays, and packet loss rates. Therefore, a key challenge is that how we can inject video codec QoE awareness to RL design and leverage neural video codec properties to resolve RL's limitations.

In this paper, we present NVC-CC, an RL-based RTC congestion control algorithm

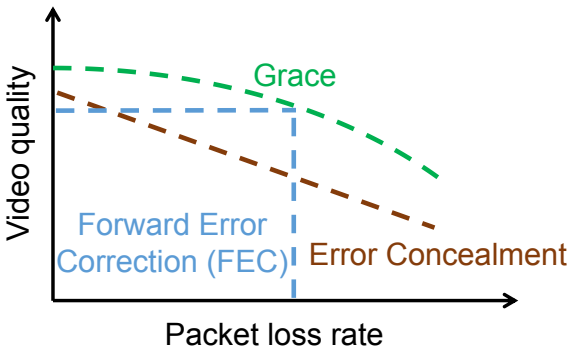


Figure 3.1: **Loss-tolerant neural video codec has slow and smooth video quality drop with increasing packet loss rate.**

which can be trained online without the help of safeguard policies by taking advantage of the NVCs' packet loss tolerance properties. Our key insight is that *the safeguard policies run along the RL model hinders the RL online learning efficiency and leveraging the loss-tolerant properties of NVC can remove RL's safeguard reliance.*

Comprehensive experiments (§3.5) on a diverse set of videos and network traces show that our NVC-aware CC running with the loss-tolerant NVC breaks loose its reliance on safeguard policies and reduces the training time by 41% compared to other prior RL-based CCs. It also boosts the mean video quality by 0.3 to 1.6dB%, lower the tail frame delay by 3 to 200ms, and reduces the video stalls by 20% to 77% in comparison with other baseline RTC CCs.

Contributions: Our work makes the following contributions.

1) We reveal the inefficiency of training in RL-based RTC congestion control solutions trained by online learning with safeguard policies and introduce the *trade-off between learning efficiency and QoE in RL training* (§3.3.2).

2) We analyze how the loss-tolerant NVCs can help improve training efficiency by allowing RL-based CCs to learn without the restriction of safeguard policies and not not hurting QoE (§3.3.3).

3) We propose NVC-CC, which, to the best of our knowledge, the first RL-based congestion control aware of and taking advantage of the loss-resilient properties of neural video codecs

(§3.4) and validate its remarkable performance gain over the state-of-the-art solutions (§3.5).

3.2 Background

To help explain NVC-CC's design, we first introduce some important concepts in real-time video communication.

3.2.1 Real-time video communication

Figure 3.2 shows a typical RTC workflow. Given a new frame and a reference frame, 1) the video codec at the sender encodes the new frame based on a target encoding bitrate estimated by the congestion control module; 2) the sender's pacer packetizes the code and paces the packets into the network according to a target sending bitrate estimated by the congestion control module; 3) the congestion control module simultaneously collects the feedback from the network and adapts the target bitrate continuously; and 4) finally the video codec at the receiver decodes the received data and reconstructs each frame.

Because lost or heavily delayed packets can affect frame decoding and introduce undesired frame quality drop, frame delay and video stalls to end users, RTC CCs need to deal with available bandwidth fluctuations with the best effort. However, it is impossible for the CC module to perfectly predict the future bandwidth changes, making heavily delayed packets and packet losses unavoidable. Thus, loss-resilient techniques and video codecs play a vital role in frame decoding especially under packet losses.

To differentiate the packet loss in network level, we define *packet loss per frame* as any packets not received before the receiver is expected to decode the frame [39].

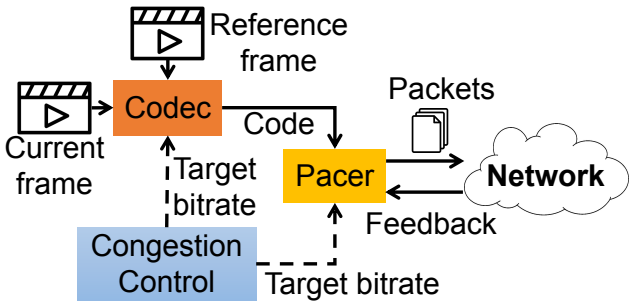


Figure 3.2: **RTC workflow.**

3.2.2 Neural video codec

Neural video codec (NVC) consists of trained neural networks instead of handcrafted logic as video encoder and decoder. Prior works in NVCs [46, 65, 94] have demonstrated comparable or even better compression efficiency comparing to traditional video codecs like H.265 [136], and VP9 [109] because they replace handcrafted heuristics in the common logical components of traditional video codes, such as motion estimation, warping, and transformative compression, with neural networks, which can learn more complex algorithms from data. Prior NVCs have also shown great generalization across a variety of video content due to their ability to be trained on a huge amount of videos. Additionally, recent studies [39, 38, 134] have discovered that NVCs have stronger loss resilience over a wider range of packet loss rates comparing to traditional loss resilience schemes (shown in Figure 3.1). To obtain stronger loss tolerance ability, the neural encoder and decoder are jointly optimized via training directly across packet loss rates. Figure 3.3 illustrates the training procedure of the loss-tolerant NVC, GRACE. Unlike traditional NVC training that assumes no data loss between the encoder and decoder, GRACE applies "random masking"–setting a fraction of randomly selected elements to zeros–to the encoder's output to simulate network packet losses. However, an important missing piece missing from prior works is that how should RL-based congestion control leverage these loss tolerant NVCs to fix its limitations.

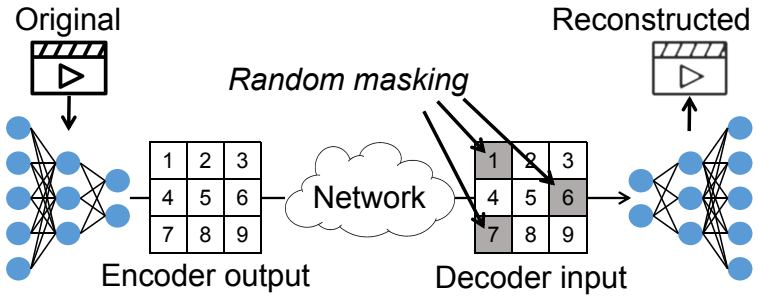


Figure 3.3: **GRACE's training procedure.**

3.3 Motivation

In this section, we will first verify the effectiveness of RL-based solutions in the field of RTC communication. Then, we are going to take OnRL [157] as an example to show that the existence of safeguard can cause learning efficiency problem and analyze the tradeoff between learning efficiency and the QoE during RL training. Lastly, we motivate how a loss-tolerant NVC can mitigate this problem.

3.3.1 RL-based CCs are promising

Recently proposed RL-based CCs have impressive improvement over traditional rule-based CCs. The RL-based CC, usually implemented by a neural network (NN), works with the transport layer and video codecs to collect states from the network environment. It then makes a decision on sending rate and broadcast the latest sending rate to the transport layer and video codes. During the training stage, its goal is to maximize a reward (typically a combination of throughput, latency, and packet loss rate).

The reasons for adopting RL in RTC CC are two-fold. Firstly, learning-based congestion controls have significant potential to self-adapt to different network conditions, eliminating the necessity for manual tuning or engineering for each specific network scenario. Contrast to traditional rule-based CCs like GCC [34] which have taken engineers years of effort to optimize to various network environments, the learning-based approach allows the adaptation

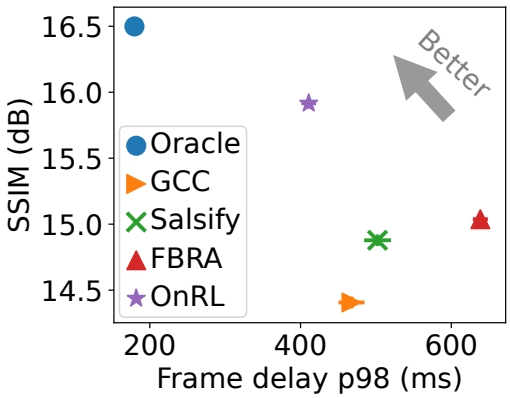


Figure 3.4: **RL-based CC has shown prospects over traditional rule-based CCs.**

to new network environments to be automatically done by machines in hours or days.

Secondly, CC problem is a sequential decision-making process, which fits well in the scope of reinforcement learning. Additionally, the neural network in deep reinforcement learning (DRL) exposes the potential and ability to learn from raw input data without the need of tedious data preprocessing and handcrafted feature engineering as long as the training network environment distribution aligns with the testing distribution.

We have conducted simulation experiments to compare an RL-based solution, OnRL, to several well-known rule-based CCs and the results do verify prior works' findings that RL-based solutions can outperform traditionally rule-based approaches when testing on network environments in the same distribution as they train on. We train and compare OnRL against rule-based CCs with a traditional video codec (e.g. H.264) on 2850 video sessions (50 diverse synthetic network traces randomly drawn from the network environment distribution in Table 3.2 and 57 videos from Table 3.1). Figure 3.4 plots the tradeoff between average frame quality and average tail frame delay among different CCs and it does show that OnRL is better than the rule-based approaches over the network environments in the same distribution that it trains on.

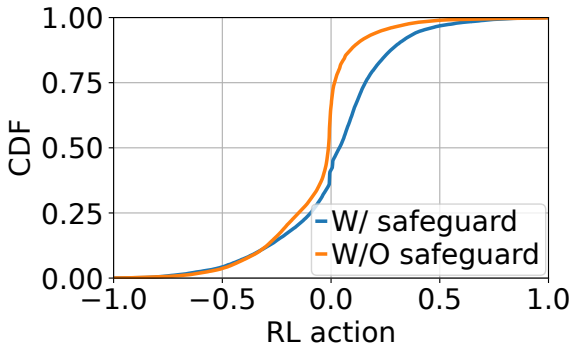


Figure 3.5: **Comparison between CDFs of sampled actions from RLs trained with and without the safeguard policy.**

3.3.2 Inefficient online RL training with safeguards

RL-based solutions are not perfect. Given that RL models learn by exploring the action space and collecting the environment feedback as mentioned in §3.1, they will inevitably conduct risky actions which may overwhelm the network capacity to cause packet delay increase or packet losses or underutilize the network available bandwidth. These risky actions eventually incur huge frame delay, poor video quality, and video stalls hurt QoE because traditional video codecs cannot decode incomplete frames and further packet retransmission might be needed.

To prevent such risky actions in RL training (especially online RL training), prior works [157, 103] detect if risky actions are made and immediately fall back to a rule-based policy to recover the system to a safe state. Only after the system is not in a risky state anymore, the RL policy mode is switched back. For example, OnRL [157] falls back to its safeguard policy, GCC, when its RL model action leads to an obvious packet jitter greater than a dynamically adjusted threshold. On the action which triggers a policy switch, it is heavily penalized so that the model is expected to learn to avoid the policy switch in the future. However, the existence of safeguard policies prevents the RL model from learning efficiently [59].

The reasons are threefold. Firstly, the existence of safeguard policies can prevent action space from being explored thoroughly and diversely during training. We verify that safeguard

policies limit RL action space exploration by comparing the distributions of sampled actions between training the RL models with and without the safeguard policy respectively on an identical network environment. Figure 3.5 shows that OnRL model with the safeguard policy spends more than 70% of time exploring the region of action space which increases the sending rates while the training without the safeguard policy balances its sampling in both action space regions that increase sending rate and regions that decrease sending rate. The diversity of actions selected by RL model with safeguard is thus much narrower than that without safeguard and RL model may potentially miss better action sequences, which in turn slows down the learning efficiency.

Secondly, the existence of safeguard policies limits RL observation space exploration. An intuitive example is to verify is that an OnRL model trained in a network environment with a deep queue network is brought to a network environment with a very short queue. Packets are dropped without apparent latency increases so its safeguard policy will not be triggered. Because RL model never sees network observations with packet losses in the training on previous network environment, it may conduct risky actions and then cause the QoE degradation until it learns from the network observations with packet loss.

Thirdly, the sampling efficiency when training RL models with safeguard policies is low. The trajectory (the sequence of tuples formed by actions, rewards, and network observations) collected by the mixture of RL model and the safeguard policy cannot be directly passed to the NN training optimizer because it violates the assumption that on-policy RL learning algorithms like PPO [131] and Markov Decision Process modeling of RL require that every action should be from the RL model. We also empirically show that safeguard in OnRL is triggered too frequently causing low sampling efficiency during the training process. For instance, Figure 3.6 shows the behavior of OnRL's sending rate within the first 30 seconds when OnRL is trained on a single network environment with respect to a 5-min video. OnRL switches into GCC for 160 times and stays in GCC for 8 seconds which are wasted in OnRL

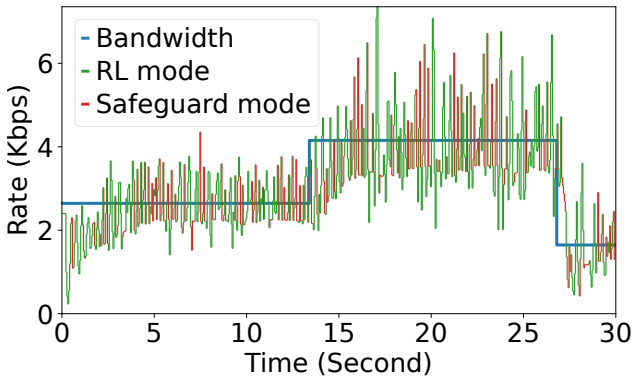


Figure 3.6: **OnRL triggers GCC-based safeguard very frequently causing low RL training sample efficiency. The network has one-way link propagation delay of 25ms, a FIFO queue with capacity of 30KB and no random loss.**

training within 30 seconds.

Therefore, there naturally exists a tradeoff between the learning efficiency of an RL model to convergence and the reward or QoE in the RL training stage. We define the learning efficiency as the number of seconds used to train an RL model to converge on a network environment. The model convergence is defined such that the RL model's testing reward on a training network environment does not vary by more than 10% if the training on this network environment continues. Following the methodology in OnRL, we use RL reward in Equation 3.2 to represent video QoE. We verify the tradeoff by training RL with and without its safeguard on a single environment and plot the training reward before model convergence and the training time used until model convergence in Figure 3.7. We sweep a wide range of safeguard sensitivity to demonstrate different degree of safeguard involvement in RL-based CC. It is clear that traditional RL-based approaches sacrifices the training efficiency in favor of less QoE degradation in RL online training. A question to ask is that: Can we remove RL reliance on safeguard policies to speed up RL training without hurt users' QoE during RL training stage using loss-tolerant NVCs?

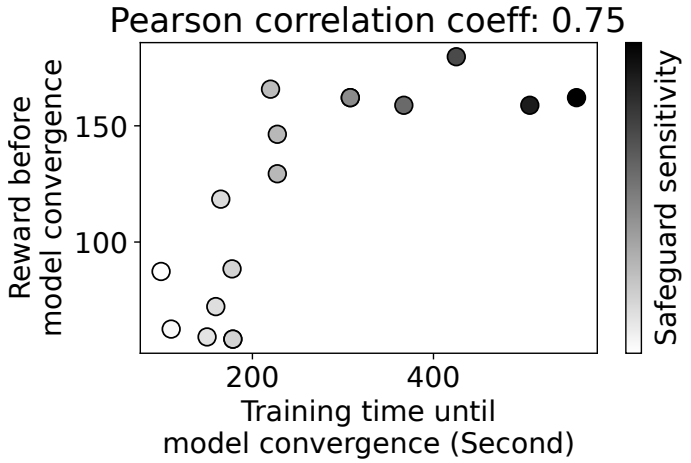


Figure 3.7: **The tradeoff between QoE (training reward) and the time RL training convergence needs. The safeguard is triggered more frequently as the color (safeguard sensitivity) darkens.**

3.3.3 How can a loss-tolerant NVC help RL-based CC training?

In this subsection, we first discuss loss-tolerant NVCs differences compared to traditional video codecs and then talk about how a loss-tolerant NVC can help RL-based CC training. The differences lie in the following two aspects:

Difference 1: loss-tolerant NVCs can translate packet delays and packet losses into video frame quality drop much smoother than traditional video codecs because they can decode an incomplete frame which is not decodable for traditional ones as shown in Figure 3.1. A takeaway is that a risky action from RL-based CC is not risky any more if the video codec is changed from traditional ones to NVC and RL-based CC can explore more actions than before.

Difference 2: A high-bitrate video frame with minor packet loss rates (i.e., less than 10%) encoded by a loss-tolerant NVC can have better quality than a low-bitrate video frame with zero packet loss does. Figure 3.8 plots the qualities of a video frame encoded by GRACE under different bitrates and frame-level packet loss rates given that the reference frame is received completely. The frame quality encoded at 1810Kbps with 10% frame data lost is approximately 2dB better than encoded at 1068Kbps with no frame data lost. However,

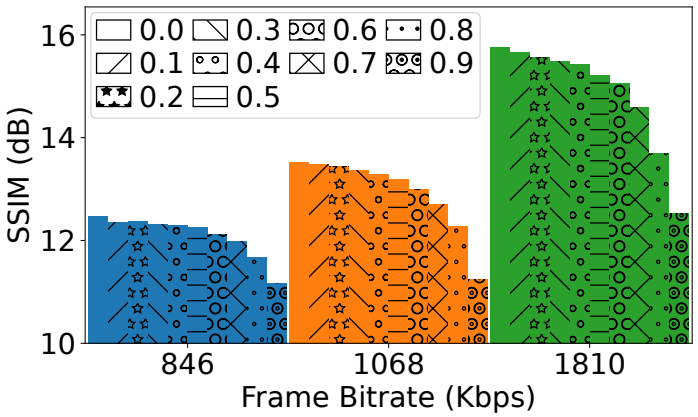
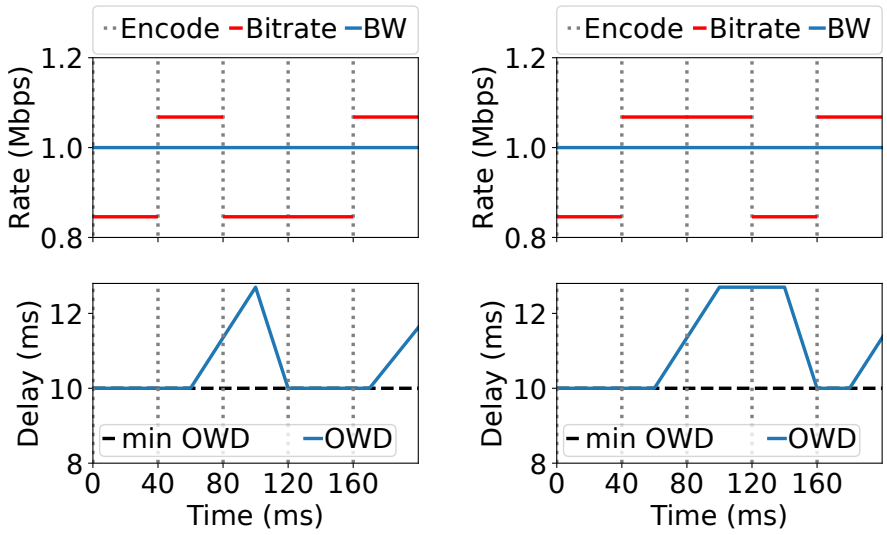


Figure 3.8: **GRACE frame quality under three encoding bitrates and various frame-level packet loss rates. Bars with the same color but different hatches represent the quality of a frame encoded at the same bitrate and received with different frame loss rates.**

because quality degradation caused by incomplete reference frames will propagate along frames. [39] suggests that a NVC state synchronization is required every 10 frames when contiguous frame losses happen. But a takeaway is that sending frames at high-bitrate with minor packet loss at frame level may yield better frame quality in between two NVC state synchronization events.

Figure 3.9 shows an example that the safeguard policy prevents the RL model from finding a better sequence of actions when the RL-based CC is trained with the loss-tolerant NVC, GRACE. In this figure, CCs is assumed to make a bitrate decision on every frame encoding event (every 40ms), and the frame data are smoothly paced out into the network before the next frame comes. As illustrated in Figure 3.9a, every time the RL chooses a bitrate overshooting the bandwidth and causes the delay inflation, the safeguard immediately takes over the control and reduces the bitrate. A possible sequence of actions that can be explored by RL without the safeguard whereas RL with safeguard will never explore due to the existence of the safeguard is in Figure 3.9b. By enduring the tail delay 12.7ms and less than 50% frame loss rate, the frame sent at 80ms is approximately 1dB in Figure 3.9b higher than that in Figure 3.9b. The average frame quality over 5 frames is about 0.2dB better.



(a) With safeguard (b) Without safeguard

Figure 3.9: **Action sequence comparison between RLs training with or without a safeguard policy. The network environment has constant bandwidth of 1.0Mbps, 10ms minimum one-way delay (OWD), and a very shallow queue of 1.5KB.**

From this example, when training with a loss-tolerant NVC, the safeguard policy is thus not necessary any more.

3.4 Design

So far we have shown that the RL-based RTC CCs look promising over traditional rule-based RTC CCs and how a loss-tolerant NVC can help the RL training if RL-based CC can be aware of it. We thus propose a new RL-based RTC CC design which leverages the loss tolerance characteristic of recent loss-tolerant NVCs via changing the reward function design.

State, action, and NN architecture: NVC-CC's observed states are designed based on those of Aurora [71] and OnRL [157]. A observed state is collected every time 50ms window, where the window length is equal to the RTCP feedback interval. It consists of the following: (i) latency gradient, the derivative of packet latency with respect to time; (ii) latency ratio, the ratio of the current time window's mean packet latency to the minimum

observed mean packet latency of any time windows in the connection's history; (iii) sending ratio, the ratio of packets sent to packets acknowledged by the receiver. To resolve the oscillation problem of sending rates of RL decision encountered in prior works, the action space is not a discrete space like OnRL's but a continuous space $[-1, 1]$. NVC-CC takes an input feature vector constructed by the states of 10 time windows and makes an action a_t to adjust the sending rate x_t for the next time window t according to Equation 3.1. γ is the throughput measured within the past 200ms. The NN architecture of NVC-CC is a small fully connected neural network with two hidden layers composed of 32 and 16 neurons respectively and tanh nonlinearity.

$$x_t = \begin{cases} x_{t-1} * a_t & a_t > 0 \\ \gamma / a_t & a_t < 0 \end{cases} \tag{3.1}$$

Reward function: Existing RL-based CCs [71, 158, 157, 150] typically compute the linear combination of throughput, packet delays, packet losses as the reward instead of directly using the feedback of video codecs because they assume a traditional video codec runs on top and cannot reflect the state of the network smoothly. For example, Aurora [71] observes network states in a time window to optimize for a reward as shown in Equation 3.2. Throughput (Tput) in kbps, latency (Lat) in second, and loss rate (Loss) are measured in a time window and $a = 120$, $b = -1000$, $c = -2000$.

$$R = \sum_i \frac{1}{n}(a \cdot \text{Tput}_i + b \cdot \text{Lat}_i + c \cdot \text{Loss}_i) \tag{3.2}$$

Instead of indirectly optimizing QoE of real-time communication session via a reward based on network-level performance, we propose to optimize a reward directly on frame quality and packet delays, enabled by the loss-tolerant video codecs. One may argue that QoE of traditional video codecs can also be used in RL-based CC training. However, because

traditional video codecs cannot always decode a frame and reflect the frame quality and delay to the sender within every decision step especially when there are packet losses or heavy packet queuing. The trial and error manner in RL model training can easily cause non-ideal decision to be made, leading to packet losses and queuing. Thus, QoE-based reward will be sparse in the RL-based CC training. The reward sparsity can slow down training or even harm the performance of the converged model. Loss-tolerant neural video codecs, on the contrary, can decode regardless of packet losses and echo the frame quality and delay back to the sender on time.

NVC-CC's reward function is defined as Equation 3.3, where $\bar{q}$ is a normalized frame quality of a video frame with respect to the frame's minimum and maximum possible frame quality, latency is packet latency (RTT) in ms, and $a = 0.1$ is the penalty coefficient on the latency term.

$$R = \sum_i (\bar{q}_i + a \cdot \text{Latency}_i)/n \tag{3.3}$$

Training: The RL model is trained online against network traces and videos in a network simulator which replay network traces with a duration of 30 seconds. The network traces are randomly generated based on the network parameters in Table 3.2. The total number of training steps is 720000. Pre-recorded video profiles are used to simulate video codec behavior and look up frame quality of GRACE encoded frames and the profiles are uniform randomly sampled during the training process.

3.5 Evaluation

Our key finds are as follows:

- **Learning efficiency:** NVC-CC running with the loss-tolerant NVC reduces the training time by 41% compared to other prior RL-based CCs.
- **Better QoE:** When testing on network traces from the same training distribution,

NVC-CC boosts the mean video quality by 0.3 to 1.6dB%, lower the tail frame delay by 3 to 200ms, and reduces the video stalls by 20% to 77% in comparison with other baseline RTC CCs.

3.5.1 Setup

Testbed implementation: We implement and compare different RTC CCs running with GRACE in a packet-level simulator, which replays network traces. All RTC CCs are run with padding enabled so that CCs can probe the available bandwidth efficiently. The simulator achieves the video encoding and decoding by replaying GRACE profiles which are collected on a server with 2 Nvidia A40 GPUs by profiling GRACE for various videos under diverse packet loss conditions. In this work, we only focus on how the congestion control algorithm affect the QoE and network-level performance so we assume the frame encoding and decoding is negligible. Optimizing the encoding and decoding time of the neural video codec is out of the scope of this work.

Baselines: We compare the NVC-CC with several baselines. These baselines include human handcrafted CCs, RL-based CCs (OnRL, Aurora), and an oracle CC.

Firstly, the traditional human handcrafted CCs include GCC, Salsify, and FBRA. GCC [34], a widely used CC in WebRTC applications, measures packet delay gradient and packet loss rate to detect network congestion and adjusts the target bitrate of video codecs at each frame. Salsify uses smoothed packet jitter to estimate the target sending rate and encodes a frame with two bitrates at a time to fast adapt the bandwidth fluctuation. FBRA sends FEC data in addition to actual video data to fast probe available bandwidth and provide protection against packet loss.

Secondly, the RL-based baselines include OnRL and Aurora. To mimic OnRL's federated learning logic that aims to provide generalizability across diverse networks, we train OnRL across diverse network environments instead of aggregating models from different users.

Unlike OnRL, Aurora is trained offline and deployed online without further finetuning.

Lastly, we add an oracle CC which knows the future bandwidth change and perfectly matches the target video bitrate to the available bandwidth. It thus serves as the optimal performance that a CC can achieve.

Videos: Our evaluation experiments reuse the test video datasets in [39], which consists of 57 videos randomly sampled from three public datasets, as shown in Table 3.1. Each video in the datasets is 10-30 seconds long. In this work, we use the same set of videos in both training and testing stages because we only focus on how network traces' diverse dynamics affect the training and testing of RL-based CCs.

Network traces: The network traces used in this work include both synthetic traces and real-world collected traces. The synthetic are synthetically generated from a network trace generator used in [150]. Each network trace is described in five dimensions including link bandwidth, minimum link round trip time (RTT), bandwidth change interval, random packet loss rate, and queue capacity with their ranges in Table 3.2. The real-world collected traces are from Pantheon dataset [151].

Quality metrics: To compare the performance of a RTC CC, in addition to network-level statistics, we report the QoE of a real-time video communication session across the following three aspects [39].

- **Video quality** of a frame is measured by structural similarity index measure (SSIM). We report SSIM in dB, computed as $-10log(1 - SSIM)$ [39, 55, 152].
- **Realtimeness** is measured by 98th percentile (p98) of frame delay (time gap between the frame's encoding and decoding).
- **Smoothness** of a video is measured by video stall (an inter-frame gap exceeding 200ms [97]). Like GRACE, we report the average number of video stalls per second and the ratio of video stall time over the entire video length.

Dataset	# of videos	Length (s)	Size	Description
Kinetics	45	450	720p 360p	Human actions and interaction with objects
Gaming	5	100	720p	Game recordings
FVC	7	140	1080p	Video calls (indoor/outdoor)
Total	57	690		

Table 3.1: Video datasets used.

Parameter	Range
Link Bandwidth (Mbps)	[0.6, 6]
Minimum link RTT (ms)	[2, 200]
Bandwidth change interval (s)	(0, 15]
Random packet loss rate	[0, 5%]
Queue capacity (packets)	[1, 100]

Table 3.2: Network parameter ranges used to generate network traces used in training and testing.

3.5.2 Learning efficiency and asymptotic performance

We first compare NVC-CC and prior RL-based CC, in terms of their *convergence speed* and *asymptotic performance* (i.e., test performance over new test environments drawn independently from the training distribution).

We train NVC-CC and baseline RL-based CCs across network environments generated from the distribution described in Table 3.2 with three different random seeds. Then we test them in new environments from the same distribution during the entire training process. Figure 3.10 plots how the validation reward guided by GRACE (defined in Equation 3.3) of each RL-based CC changes over training time. NVC-CC converges to the best test reward of 0.859, which is 50% and 26% better than that of Aurora and that of OnRL respectively. It takes NVC-CC 78 minutes to converge to the best reward, which is approximately 41% faster than both Aurora and OnRL.

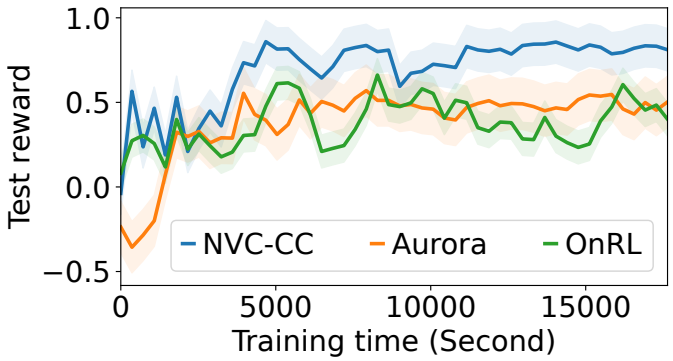


Figure 3.10: **NVC-CC training ramps up faster and converges to a better test reward than the baseline RL-based CCs.**

3.5.3 Performance evaluation on synthetic network traces

We then compare NVC-CC and baseline CCs on the network traces which are drawn from the training distribution but unseen in the training process. The performance is broke down in terms of the real-time video communication QoE metrics mentioned in §3.5.1. We run them with GRACE on 2850 video sessions. These video sessions are composed by 50 network environments randomly sampled from the distribution in Table 3.2 and 57 videos described in Table 3.1. Their QoE breakdown is shown in Figure 3.11. NVC-CC outperforms the traditional rule-based RTC CCs by more than 1.6dB in average frame quality (SSIM). It outperforms OnRL and Aurora by at least 0.3dB. In terms of realtimeness, it leads the rule-based CCs by 3 to 200ms and leads Aurora and OnRL by 200ms. NVC-CC causes fewer video stalls than most of rule-based CCs and RL-based CCs except GCC which is known to be conservative. Salsify is outperformed by NVC-CC because GRACE is not a functional video codec Salsify requires and its fast bandwidth adaptation mechanism does not work well. FBRA is outperformed by the most of other approaches because it often overshoots the network capacity by abusing FEC data too much.

We also break down the CC's performance from the network level. From Table 3.3, it is clear that NVC-CC is able to aggressively consume the available bandwidth by trading off minor degree of packet losses as it is aware of loss-tolerant NVC performance via the reward

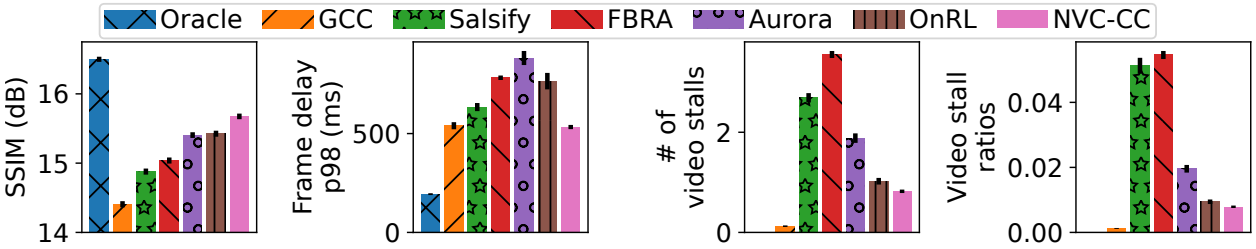


Figure 3.11: **QOE breakdown when evaluating on synthetic network traces from the training distribution.**

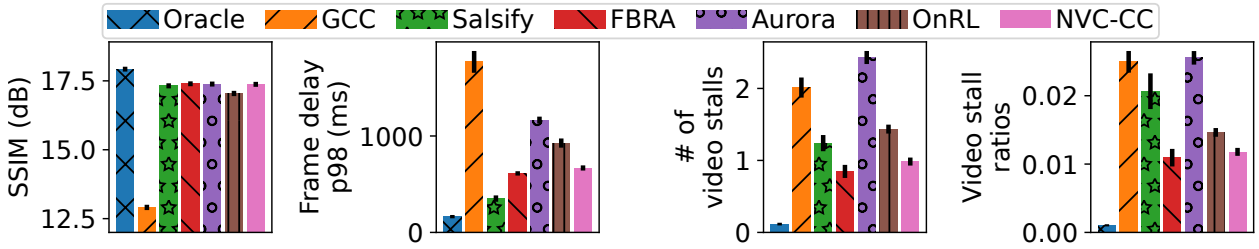
CC	Tput (Mbps)	Delay p98 (ms)	Loss (%)
Oracle	3.17	113	0.1
GCC	1.22	287	1.5
Salsify	1.72	272	5.6
FBRA	2.19	478	18.9
Aurora	2.59	376	4.6
OnRL	2.12	325	2.4
NVC-CC	2.55	355	3.5

Table 3.3: Network-level performance breakdown of Figure 3.11

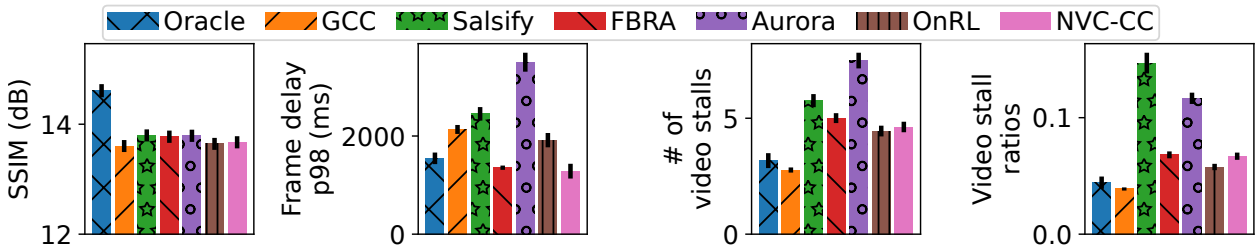
in Equation 3.3.

3.5.4 Performance evaluation on real-world network traces

In addition to evaluating the CCs on the network traces from the training distribution, we test NVC-CC's generalizability on network traces outside the training distribution by evaluating them real-world collected traces from Pantheon dataset [151]. These traces contain two major network settings–ethernet and cellular and they were collected on 10 different links among nodes globally. Each CC is the tested on 570 video sessions and the resulted QoE breakdown are shown in Figure 3.12a and Figure 3.12b respectively. Although NVC-CC still outperforms prior RL-based CCs along all four QoE metrics in both network settings, its benefits over traditional rule-based CCs are marginal, indicating that NVC-CC's generalizability on network traces outside its training distribution still needs to improve.



(a) Ethernet



(b) Cellular

Figure 3.12: **QOE breakdown when evaluating on real-world network traces.**

3.6 Discussion

While the preliminary results show early promise of NVC-CC, there are still limitations in its methodology and future steps needed to be done.

Video and codec dependency: Because NVC-CC's reward used in training relies on the frame quality of a video frame, there naturally exists a dependency between the trained NVC-CC model and the video codec as well as the training videos. The dependency on video codec indicates that a different NVC-CC model should be used when the loss-tolerant NVC is updated. To address the dependency on training videos, more research effort needs to be invested in measuring and improving NVC-CC RL model's generalization over videos with diverse content.

Sim-to-real generalization: Similar to other RL models trained in a simulated environment and deployed in the real world, NVC-CC also faces the simulation to real world generalization problem. Even though NVC-CC is trained using online RL training, our experiments are conducted by replaying network traces in a network simulator. Thus, an

important future step to investigate is to measure and improve NVC-CC's generalizabilty over real-world network environments.

Fairness: On the Internet, many different congestion control protocols need to interact, and ours can possibly behave unfairly in some scenarios. For example, being aware of loss-tolerant NVC allows NVC-CC to be aggressive in increasing packet sending rate and unaffected by minor packet drops. It is possible that NVC-CC takes up the majority of link capacity and forces traditional TCP to back off quickly.

Scalability: Due to the reliance on the performance of loss-tolerant NVCs, NVC-CC also inherits some limitations from NVCs. For instance, GRACE is not optimized enough to run at 30 fps on resource-constrained devices that barely sustain classic video codec. Thus, NVC-CC cannot be trained to support 30 fps real time streaming on such devices. Another example is that GRACE focuses on unicast video communication instead of multi-party video conferencing. Therefore, the performance of NVC-CC in multi-party video conferencing scenario is still unknown.

3.7 Conclusion

We present NVC-CC, an RL-based congestion control algorithm for real time video communication applications. For the first time, we analyze the tradeoff between quality of experience during RL training stage and RL model convergence speed. We reveal that the commonly used safeguard policies used to prevent risky actions and recover the networked system to safe states can cause sampling efficiency and slow down the RL model convergence. NVC-CC addresses the training efficiency problem by training RL model on top of loss-tolerant neural video codec without the need of safeguard policies. Our evaluation shows that NVC-CC's training with loss-tolerant neural video codec improves RL training efficiency and achieves better quality of experience in various network environments and video workloads.

CHAPTER 4

CONCLUSION

4.1 Contributions

This thesis contributes by proposing a universal training framework via leveraging network domain knowledge to reweight the reward observed in RL training. The training framework aims to overcome the limitations faced by for DRL-based rate adaptations. These limitations include: 1) poor generalizability across diverse network environments, and 2) lack of awareness of the user-perceived quality of experience. To mitigate the above limitations and instantiate the training framework, we have proposed two exemplifications, GENET and NVC-CC, which using two different network domain knowledge respectively (rule-based baselines and video codecs).

GENET adopts the philosophy of curriculum learning by gradually feeding more "difficult" environments to the training rather than choosing them uniformly at random. It leverages the intuition that further training a RL model in a network environment tends to bring substantial improvement if the RL model performs significantly worse in the environment than the rule-based baselines. GENET automatically searches for such environments and iteratively promotes them to training. In three case studies (adaptive video streaming, congestion control, and load balancing), GENET produces RL policies that outperform both regularly trained RL policies and traditional baselines inside the training distribution. GENET improves asymptotic performance by 8–25% for ABR, 14–24% for CC, 15% for LB, compared with traditional RL training methods. Although not specifically optimized, GENET also improves the model performance when tested outside the training distribution.

NVC-CC is an RL-based congestion control algorithm which takes the advantage of loss resilience properties of recent neural video codecs and incorporates the codec feedback in the training reward function. Because of its awareness of loss tolerance in neural video

codec (trained with reward including feedback from both the video codec and the network), NVC-CC overcomes the reliance on safeguard policies and improves the training efficiency. Through extensive evaluation on various videos and network traces in a simulated environment, NVC-CC reduces the training time by 41% compared to other prior RL-based CCs when both NVC-CC and prior RL-based CCs are trained with the loss-tolerant neural video codec. It also boosts the mean video quality by 0.3 to 1.6dB%, lower the tail frame delay by 3 to 200ms, and reduces the video stalls by 20% to 77% in comparison with other baseline RTC CCs such as GCC, Salsify, and FBRA.

4.2 Future work

Four potential directions in machine learning for networking and systems can be explored in the future.

Robustness of RL in dynamic action space: The state-of-art RL-based approaches for network and systems are designed for static action space over time. However, this assumption does not always hold in real life. For instance, an RL-based solution for load balancing requires to choose from a fixed set of machines. If any machine in the action space used for training crashes due to hardware fault or the action space grows by adding new machines, it is highly possible that the RL model makes unwise choices like keeping choosing the faulty machine or ignoring the newly added machines. Therefore, an important missing piece in prior works is that how robust the RL-based approach is with respect to a dynamic action space. There are many possible solutions like masking off invalid actions [66], completely retraining the deployed RL model on new action space, finetuning the last layer of the RL model, or even training RL models with a dynamic action space [36]. My research will systematically explore and compare these potential solutions and try to improve the robustness of RL-based approaches on dynamic action space.

Trace generation using generative neural networks: Current RL-based approaches

especially those trained offline in a network simulator rely heavily on replaying network traces during their training stage. These network traces can be classified into either synthetic traces generated by heuristics or real-world collected traces. Synthetic traces are easy to generate but are known to lack real-world network dynamics and fidelity. Real-world traces overcome the drawbacks of synthetic traces are nontrivial and time consuming to collect and tend to involve with privacy and legal issues. My future research direction is to leverage the latest (controllable) generative adversarial network (GAN) [92], which are trained on real-world network traces, to synthesize network traces for RL training. The intuition is that generative neural networks can hold the fidelity and dynamics of real-world collected traces but generate distinct traces from their training data.

Running time optimization for RL-based approaches and neural video codecs: Existing RL-based solutions for networking and systems typically uses the neural network models with merely 2 to 3 fully connected layers because only shallow models which have low running time on CPU-only hardware settings can satisfy the millisecond-level decision granularity required by modern systems and network. However, according to the experience in the field of computer vision and natural language processing, large and deep models show amazing performance and accuracy. It indicates that deeper and more complex neural networks still have huge potential in the field of networking if their running time can be further optimized. Therefore, another future research direction is explore more architectures for neural networks in systems and networking and optimize their running time for high performance system and high speed networks.

Running time optimization for neural video codecs should also be conducted in the future. The state-of-art neural video codecs require to be run on GPU and cannot keep up with more than the frame rate of 25 FPS in video streaming applications [39]. Therefore, more effort should be spent on the integration of neural video codecs with current WebRTC framework and its speed-up.

Safeguard policy experience reuse: Demonstrate in §3.3, safeguard policy can often dominate the networked system and hinder RL exploration of the action space, causing poor training efficiency. A key question to ask is how to leverage the experience collected by the safeguard policy to help RL model training so it will not be wasted. Although prior works have proposed possible solutions including translating a safeguard policy into a neural network which can be co-trained with RL model via feature fusion [158] and rewarding environment selection by comparing the performance of rule-based safeguard and that of RL model [150], they still fall short of complicated training process and poor decision granularity. Therefore, new methodologies should be investigated in the future.

BIBLIOGRAPHY

[1] WebRTC and IoT Applications. https://rtcweb.in/webrtc-and-iot-applications/.

[2] Aurora implementation. https://github.com/PCCproject/PCC-RL.

[3] Yoda: Video analytics workload benchmark for performance clarity. https://github.com/zxxia/benchmarking.

[4] DDS streaming pipeline codebase. https://github.com/KuntaiDu/dds.

[5] OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. https://github.com/openai/gym.

[6] OpenNetLab. https://opennetlab.org/.

[7] Park implementation. https://github.com/park-project/park.

[8] Pensieve implementation. https://github.com/hongzimao/pensieve.

[9] Workshop on ML for Systems at NeurIPS 2018. http://mlforsystems.org/neurips2018/, 2018.

[10] Workshop on ML for Systems at NeurIPS 2019. http://mlforsystems.org/neurips2019/, 2019.

[11] Workshop on ML for Systems at ISCA 2019. http://mlforsystems.org/isca2019/, 2019.

[12] WebRTC: Enabling Collaboration Augmented Reality App. https://arvrjourney.com/webrtc-enabling-collaboration-cebdd4c9ce06?gi=e19b1c0f65c0, 2020. Accessed: 2024-08-12.

[13] Open Source Cloud Gaming with WebRTC. https://webrtchacks.com/open-source-cloud-gaming-with-webrtc/, 2020. Accessed: 2024-08-12.

[14] Workshop on ML for Systems at NeurIPS 2020. http://mlforsystems.org/, 2020.

[15] WebRTC Cloud Gaming: Unboxing Stadia. https://webrtc.ventures/2021/02/webrtc-cloud-gaming-unboxing-stadia/, 2021. Accessed: 2024-08-12.

[16] Features of WebRTC VR Streaming. https://flashphoner.com/features-of-webrtc-vr-streaming/, 2021. Accessed: 2024-08-12.

[17] WebRTC: Real-Time Communication for the Internet of Things. https://mobidev.biz/blog/webrtc-real-time-communication-for-the-internet-of-things, 2024. Accessed: 2024-08-12.

[18] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 632–647, 2020.

[19] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29 (13):1608–1639, 2010.

[20] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

[21] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[22] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[23] Ahmed Badr, Ashish Khisti, Wai-tian Tan, Xiaoqing Zhu, and John Apostolopoulos. Fec for voip using dual-delay streaming codes. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[24] Mihovil Bartulovic, Junchen Jiang, Sivaraman Balakrishnan, Vyas Sekar, and Bruno Sinopoli. Biases in data-driven networking, and what to do about them. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 192–198, 2017.

[25] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

[26] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, pages 41–48, 2009.

[27] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

[28] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX*

Symposium on Networked Systems Design and Implementation (NSDI 22), pages 119–135, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL `https://www.usenix.org/conference/nsdi22/presentation/bhardwaj`.

[29] Niklas Blum, Serge Lachapelle, and Harald Alvestrand. WebRTC: Real-time communication for the open web platform. *Communications of the ACM*, 64(8):50–54, 2021.

[30] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3722–3731, 2017.

[31] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, 1994.

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[33] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[34] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[35] Fabrizio Carpi, Christian Häger, Marco Martalò, Riccardo Raheli, and Henry D Pfister. Reinforcement learning for channel coding: Learned bit-flipping decoding. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 922–929. IEEE, 2019.

[36] Yash Chandak, Georgios Theocharous, Blossom Metevier, and Philip Thomas. Reinforcement learning when all actions are not always available. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3381–3388, 2020.

[37] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. *Advances in Neural Information Processing Systems*, 30:1002–1012, 2017.

[38] Bo Chen, Zhisheng Yan, Yinjie Zhang, Zhe Yang, and Klara Nahrstedt. Lifter: Unleash learned codecs in video streaming with loose frame referencing. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 533–548, 2024.

[39] Yihua Cheng, Ziyi Zhang, Hanchen Li, Anton Arapin, Yue Zhang, Qizheng Zhang, Yuhan Liu, Kuntai Du, Xu Zhang, Francis Y Yan, et al. Grace: Loss-resilient real-time video through neural codecs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 509–531, 2024.

[40] Sandeep Chinchali, Pan Hu, Tianshu Chu, Manu Sharma, Manu Bansal, Rakesh Misra, Marco Pavone, and Sachin Katti. Cellular network traffic scheduling with deep reinforcement learning. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.

[41] Wen-Jeng Chu and Jin-Jang Leou. Detection and concealment of transmission errors in h. 261 images. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(1):74–84, 1998.

[42] Cisco. Annual internet report (2018–2023). `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`. Accessed: 2024-08-12.

[43] Chameleon cloud server. Chameleon cloud server. `https://www.chameleoncloud.org/`.

[44] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[45] Federal Communications Commission. Measuring Broadband America. `https://www.fcc.gov/general/measuring-broadband-america`.

[46] Mallesham Dasari, Kumara Kahatapitiya, Samir R Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, 2022.

[47] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *arXiv preprint arXiv:2012.02096*, 2020.

[48] Arnaud Dethise, Marco Canini, and Srikanth Kandula. Cracking open the black box: What observations can tell us about reinforcement learning agents. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pages 29–36, 2019.

[49] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[50] Kurt Driessens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

[51] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. Server-driven video streaming for deep learning inference. In *Proceedings of the ACM Special Interest Group on Data Communication*, 2020.

[52] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 305–318, 2021.

[53] Jeongyoon Eo, Zhixiong Niu, Wenxue Cheng, Francis Y. Yan, Rui Gao, Jorina Kardhashi, Scott Inglis, Michael Revow, Byung-Gon Chun, Peng Cheng, and Yongqiang Xiong. OpenNetLab: Open platform for RL-based congestion control for real-time communications. In *6th Asia-Pacific Workshop on Networking (APNet 2022)*, 2022.

[54] Joyce Fang, Martin Ellis, Bin Li, Siyao Liu, Yasaman Hosseinkashi, Michael Revow, Albert Sadovnikov, Ziyuan Liu, Peng Cheng, Sachin Ashok, et al. Reinforcement learning for bandwidth estimation and congestion control in real-time communications. *arXiv preprint arXiv:1912.02222*, 2019.

[55] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-Latency network video through tighter integration between a video codec and a transport protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 267–282, 2018.

[56] Peter I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[57] Mohammad Ghavamzadeh, Marek Petrik, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. *Advances in Neural Information Processing Systems*, 29:2298–2306, 2016.

[58] Tomer Gilad, Nathan H Jay, Michael Shnaiderman, Brighten Godfrey, and Michael Schapira. Robustifying network protocols with adversarial examples. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 85–92, 2019.

[59] Shangding Gu, Laixi Shi, Yuhao Ding, Alois Knoll, Costas Spanos, Adam Wierman, and Ming Jin. Enhancing efficiency of safe reinforcement learning via sample manipulation. *arXiv preprint arXiv:2405.20860*, 2024.

[60] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[61] Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In *International Conference on Machine Learning*, pages 2535–2544. PMLR, 2019.

[62] Ameer Haj-Ali, Nesreen K Ahmed, Ted Willke, Joseph Gonzalez, Krste Asanovic, and Ion Stoica. A view on deep reinforcement learning in system optimization. *arXiv preprint arXiv:1908.01275*, 2019.

[63] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

[64] Marie Hopkins. Live sports virtual reality broadcasts: Copyright and other protections. *Duke L. & Tech. Rev.*, 16:141, 2017.

[65] Zhihao Hu, Guo Lu, and Dong Xu. Fvc: A new framework towards deep video compression in feature space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1502–1511, 2021.

[66] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

[67] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[68] Ismaeil Ismaeil, Shahram Shirani, Faouzi Kossentini, and Rabab Ward. An efficient, similarity-based error concealment method for block-based coded images. In *Proceedings 2000 International Conference on Image Processing (Cat. No. 00CH37101)*, volume 3, pages 388–391. IEEE, 2000.

[69] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, 1988.

[70] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph Gonzalez. Scaling video analytics systems to large camera deployments. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 9–14, 2019.

[71] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[72] Junchen Jiang. *Enabling Data-Driven Optimization of Quality of Experience in Internet Applications*. PhD thesis, Carnegie Mellon University, 2017.

[73] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, and Hui Zhang. Via: Improving internet telephony call quality

using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 286–299, 2016.

[74] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. CFA: A practical prediction system for video qoe optimization. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 137–150, 2016.

[75] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 393–406, 2017.

[76] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew A Chien. Networked cameras are the new big data clusters. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 1–7, 2019.

[77] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661. PMLR, 2016.

[78] Ingemar Johansson and Zaheduzzaman Sarker. Self-Clocked Rate Adaptation for Multimedia. RFC 8298, December 2017. URL `https://www.rfc-editor.org/info/rfc8298`.

[79] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

[80] Daniel Kahneman, Jack L Knetsch, and Richard H Thaler. Anomalies: The endowment effect, loss aversion, and status quo bias. *Journal of Economic Perspectives*, 5(1):193–206, 1991.

[81] Jaeyeon Kang, Seoung Wug Oh, and Seon Joo Kim. Error compensation framework for flow-guided video inpainting. In *European Conference on Computer Vision*, pages 375–390. Springer, 2022.

[82] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. Verifying deep-RL-driven systems. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, pages 83–89, 2019.

[83] Rawal Khirodkar, Donghyun Yoo, and Kris M Kitani. Vadra: Visual adversarial domain randomization and augmentation. *arXiv preprint arXiv:1812.00491*, 2018.

[84] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

[85] M. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, volume 23, 2010.

[86] Sunil Kumar, Liyang Xu, Mrinal K Mandal, and Sethuraman Panchanathan. Error resiliency schemes in H. 264/AVC standard. *Journal of Visual Communication and Image Representation*, 17(2):425–450, 2006.

[87] Peter Lambert, Wesley De Neve, Yves Dhondt, and Rik Van de Walle. Flexible macroblock ordering in h. 264/avc. *Journal of Visual Communication and Image Representation*, 17(2):358–375, 2006.

[88] Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR, 2019.

[89] Mathias Lecuyer, Joshua Lockerman, Lamont Nelson, Siddhartha Sen, Amit Sharma, and Aleksandrs Slivkins. Harvesting randomness to optimize distributed systems. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 178–184, 2017.

[90] Insoo Lee, Seyeon Kim, Sandesh Sathyanarayana, Kyungmin Bin, Song Chong, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. R-FEC: RL-based FEC Adjustment for Better QoE in WebRTC. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 2948–2956, 2022.

[91] Yueheng Li, Zicheng Zhang, Hao Chen, and Zhan Ma. Mamba: Bringing Multi-Dimensional ABR to WebRTC. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9262–9270, 2023.

[92] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020.

[93] Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *arXiv preprint arXiv:1810.12429*, 2018.

[94] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. DVC: An End-to-end Deep Video Compression Framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[95] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

[96] David JC MacKay and Radford M Neal. Near shannon limit performance of low density parity check codes. *Electronics Letters*, 33(6):457–458, 1997.

[97] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 229–244, 2021.

[98] Mohammadhossein Malmir, Josip Josifovski, Noah Klarmann, and Alois Knoll. Robust sim2real transfer by learning inverse dynamics of simulated systems. In *2nd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics*, 2020.

[99] H Mao, S Chen, D Dimmery, S Singh, D Blaisdell, Y Tian, M Alizadeh, and E Bakshy. Real-world video adaptation with reinforcement learning. arxiv 2020. *arXiv preprint arXiv:2008.12858*.

[100] Hongzi Mao. Pensieve collected data. `https://www.dropbox.com/sh/ss0zs1lc4cklu3u/AAB-8WC3cHD4PTtYT0E4M19Ja?dl=0`.

[101] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.

[102] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, et al. Park: An open platform for learning-augmented computer systems. *Advances in Neural Information Processing Systems*, 32, 2019.

[103] Hongzi Mao, Malte Schwarzkopf, Hao He, and Mohammad Alizadeh. Towards safe online reinforcement learning in computer systems. In *NeurIPS Machine Learning for Systems Workshop*, 2019.

[104] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288. 2019.

[105] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[106] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization. In *Conference on Robot Learning*, pages 1162–1176. PMLR, 2020.

[107] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[108] Matthew K Mukerjee, David Naylor, Junchen Jiang, Dongsu Han, Srinivasan Seshan, and Hui Zhang. Practical, real-time centralized control for cdn-based live video delivery. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 311–324, 2015.

[109] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. The latest open-source video codec VP9-an overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*, pages 390–393. IEEE, 2013.

[110] Marcin Nagy, Varun Singh, Jörg Ott, and Lars Eggert. Congestion control using FEC for conversational multimedia communication. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 191–202, 2014.

[111] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574, 2016.

[112] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020.

[113] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[114] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[115] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[116] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[117] Lerrel Pinto, James Davidson, and Abhinav Gupta. Supervision via competition: Robot adversaries for learning tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1601–1608. IEEE, 2017.

[118] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep RL: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

[119] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032*, 2019.

[120] Devdeep Ray, Connor Smith, Teng Wei, David Chu, and Srinivasan Seshan. SQP: Congestion control for low-latency interactive video streaming. *arXiv preprint arXiv:2207.11857*, 2022.

[121] Zhipeng Ren, Daoyi Dong, Huaxiong Li, and Chunlin Chen. Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2216–2226, 2018.

[122] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 114–118, 2013.

[123] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P Lillicrap, and Greg Wayne. Experience replay for continual learning. *arXiv preprint arXiv:1811.11682*, 2018.

[124] Noga H. Rotman, Michael Schapira, and Aviv Tamar. Online safety assurance for deep reinforcement learning. *arXiv preprint arXiv:2010.03625*, 2020.

[125] Michael Rudow, Francis Y Yan, Abhishek Kumar, Ganesh Ananthanarayanan, Martin Ellis, and KV Rashmi. Tambur: Efficient loss recovery for videoconferencing via streaming codes. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 953–971, 2023.

[126] Fereshteh Sadeghi and Sergey Levine. CAD2R: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[127] Arun Sankisa, Arjun Punjabi, and Aggelos K Katsaggelos. Video error concealment using deep neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 380–384. IEEE, 2018.

[128] Michael Schaarschmidt. End-to-end deep reinforcement learning in computer systems. Technical report, University of Cambridge, Computer Laboratory, 2020.

[129] Michael Schaarschmidt, Kai Fricke, and Eiko Yoneki. Wield: Systematic reinforcement learning with progressive randomization. *arXiv preprint arXiv:1909.06844*, 2019.

[130] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in Psychology*, 4:313, 2013.

[131] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[132] Junyang Shi, Mo Sha, and Xi Peng. Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021.

[133] Felipe Leno Da Silva and Anna Helena Reali Costa. Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1026–1034, 2018.

[134] Vibhaalakshmi Sivaraman, Pantea Karimi, Vedantha Venkatapathy, Mehrdad Khani, Sadjad Fouladi, Mohammad Alizadeh, Frédo Durand, and Vivienne Sze. Gemino: Practical and robust neural compression for video conferencing. *arXiv preprint arXiv:2209.10507*, 1, 2022.

[135] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[136] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[137] Lalith Suresh, Marco Canini, Stefan Schmid, and Anja Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 513–527, 2015.

[138] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[139] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[140] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 969–977, 2018.

[141] Joanne Truong, Sonia Chernova, and Dhruv Batra. Bi-directional domain adaptation for sim2real transfer of embodied navigation agents. *IEEE Robotics and Automation Letters*, 6(2):2634–2641, 2021.

[142] Cameron Voloshin, Hoang M Le, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*, 2019.

[143] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, 2018. doi:10.1109/MNET.2017.1700200.

[144] Yi Wang, Xiaoqiang Guo, Feng Ye, Aidong Men, and Bo Yang. A novel temporal error concealment framework in H. 264/AVC. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2013.

[145] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. Bridging the edge-cloud barrier for real-time advanced vision analytics. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[146] Daphna Weinshall, Gad Cohen, and Dan Amir. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *International Conference on Machine Learning*, pages 5238–5246. PMLR, 2018.

[147] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[148] Keith Winstein and Hari Balakrishnan. TCP ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review*, 43(4):123–134, 2013.

[149] World Wide Web Consortium. Web Real-Time Communication (WebRTC) 1.0: Real-Time Communication Between Browsers. `https://www.w3.org/TR/webrtc/`, 2015.

[150] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 397–413, 2022.

[151] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for Internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, Boston, MA, July 2018. USENIX Association. ISBN 978-1-939133-01-4. URL `https://www.usenix.org/conference/atc18/presentation/yan-francis`.

[152] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020.

[153] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning *in situ*: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL `https://www.usenix.org/conference/nsdi20/presentation/yan`.

[154] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[155] Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*, pages 1–5, 2015.

[156] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[157] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. OnRL: improving mobile video telephony via online reinforcement learning. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[158] Huanhuan Zhang, Anfu Zhou, Yuhan Hu, Chaoyue Li, Guangping Wang, Xinyu Zhang, Huadong Ma, Leilei Wu, Aiyun Chen, and Changhui Wu. Loki: improving long tail performance of learning-based real-time video adaptation by fusing rule-based models. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 775–788, 2021.

[159] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[160] Jie Zhou, Bo Yan, and Hamid Gharavi. Efficient motion vector interpolation for error concealment of H. 264/AVC. *IEEE Transactions on Broadcasting*, 57(1):75–80, 2010.

[161] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 258–271, 2021.

[162] Xiaoqing Zhu, Rong Pan *, Michael A. Ramalho, and Sergio Mena de la Cruz. Network-Assisted Dynamic Adaptation (NADA): A Unified Congestion Control Scheme for Real-Time Media. RFC 8698, February 2020. URL `https://www.rfc-editor.org/info/rfc8698`.

[163] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

APPENDIX A

GENET

A.1 Details of RL implementation

The input of RL algorithm consists of a space of configurations, an initial policy parameters and predefined total number of iterations to train. The space of configurations is constructed by ranges of environment configurations. Each range is marked by the configuration's min and max values. Within a training iteration, each dimension of the space of configurations is uniformly sampled to create K configurations. For each configuration, N random environments are created. Thus, rollouts are collected by running the policy on total $K \times N$ environments to update the policy. When the policy is updated for the predefined number of iterations, the RL algorithm stops training and outputs a trained policy.

Algorithm 1 Traditional Reinforecment Learning (RL)

Input: Ω: space of configurations, θ: initial policy parameters, N_{iters}: # of iterations
Output: θ: returned policy parameters

for i from 1 to N_{iters} **do**
  $\Phi_{rand} \leftarrow \emptyset$
  for 1 **to** K **do** ▷ K: # configs per iteration
    $p_i \sim Random(\Omega)$ ▷ Uniformly sampled config in Ω
    for 1 **to** N **do** ▷ N: # random envs per config
      $E \leftarrow S(p_i)$ ▷ Create a simulated env by p_i
      rollout $\phi \sim \pi_\theta(\cdot; E)$ ▷ Rollout policy π_θ on E
      $\Phi_{rand} \leftarrow \Phi_{rand} \cup \phi$
    end for
  end for
  with Φ_{rand} update:
  $\theta \leftarrow \theta + \nu \bigtriangledown_\theta J(\pi_\theta)$ ▷ Gradient update with rate ν
end for
return θ

ABR Parameter	RL1	RL2	RL3	Default	Original
Max playback buffer (s)	[2, 10]	[2, 50]	[2, 100]	60	60
Video chunk length (s)	[1, 4]	[1, 6]	[1, 10]	4	4
Min link RTT (ms)	[20, 30]	[20, 220]	[20, 1000]	80	80
Video length (s)	[40, 45]	[40, 200]	[40, 400]	196	196
Bandwidth change interval (s)	[2, 2]	[2, 20]	[2, 100]	5	
Max link bandwidth (Mbps)	[2, 5]	[2, 100]	[2, 1000]	5	

Table A.1: Parameters in ABR simulation. Colored rows show the configurations (and their ranges) used in the simulator in the original paper. The synthetic trace generator is described in §A.2.

CC Parameter	RL1	RL2	RL3	Default	Original
Maximum link bandwidth (Mbps)	[0.5, 7]	[0.4, 14]	[0.1, 100]	3.16	[1.2, 6]
Minimum link RTT (ms)	[205, 250]	[156, 288]	[10, 400]	100	[100, 500]
Bandwidth change interval (s)	[11, 13]	[8, 3]	[0, 30]	7.5	
Random loss rate	[0.01, 0.014]	[0.007, 0.02]	[0, 0.05]	0	[0, 0.05]
Queue (packets)	[2, 6]	[2, 11]	[2, 200]	10	[2, 2981]

Table A.2: Parameters in CC simulation. Colored rows show the configurations (and their ranges) used in the simulator in the original paper. The synthetic trace generator is described in §A.2. The range of RL1 is defined as 1/9 of the range of RL3 and the range of RL2 is defined as 1/3 of RL3. The CC parameters shown here for RL1 and RL2 are example sets.

A.2 Trace generator logic

ABR: For the simulation in ABR, the link bandwidth trace has the format of [timestamp (s), throughput (Mbps)]. Our synthetic trace generator includes 4 parameters: minimum BW (Mbps), maximum BW (Mbps), BW changing interval (s), and trace duration (s). Each timestamp represents one second with a uniform [-0.5, 0.5] noise. Each throughput follows a uniform distribution between [min BW, max BW]. The BW changing interval controls how often throughput change over time, with uniform [1, 3] noise. Trace duration represents the total time length of the current trace.

CC: The trace generator in the CC simulation takes 6 inputs: maximum BW (Mbps), BW

LB Parameter	RL1	RL2	RL3	Default	Original
Service rate	[0.1, 2]	[0.1, 5]	[0.1, 10]	[0.5, 1.0, 2.0]	[2, 4]
Job size (byte)	[100, 200]	[100, 10^3]	[1, 10^4]	2000	[100, 1000]
Job interval (ms)	[0.01, 0.05]	[0.01, 0.1]	[0.1, 1]	0.1	0.2
Number of jobs	[10, 100]	[10, 1000]	[10, 5000]	2000	1000
Queue shuffled probability	[0.1, 0.2]	[0.1, 0.5]	[0.1, 1]	0.5	

Table A.3: Parameters in LB simulation. Colored rows show the configurations (and their ranges) used in the simulator in the original paper. The synthetic trace generator is described in §A.2.

changing interval (s), link one-way latency (ms), queue size (packets), link random loss rate, delay noise (ms), and duration (s). It outputs a series of timestamps with 0.1s step length and dynamic bandwidth series. Each bandwidth value is drawn from a uniform distribution of range [1, max BW] Mbps. The BW changing interval allows bandwidth to change every certain seconds. The link one-way latency is used to simulate packet RTT. The queue size simulates a single queue in a sender-receiver network. Link random loss rate determines the chance of random packet loss in the network. Delay noise determines how large a Gaussian noise is added to a packet. The trace duration is determined by the duration input.

LB: We use the similar workload traces generator as the Park [7] project, where jobs arrive according to a Poisson process, and the job sizes follow a Pareto distribution with parameters [shape, scale]. In the simulation, all servers process jobs from their queues at identical rates.

A.3 Details of Figure 2.4

Trace sets in Figure 2.4 was generated by two configurations. For trace set X, we used BW range: 0–5Mbps, BW changing frequency: 0–2s. For trace set Y, we used BW range: 0–10Mbps, BW changing frequency: 4–15s. As a motivation example, each trace set contains 20 traces to show the testing reward trend.

Algorithm 2 GENET training framework

Input: Ω: uniform configuration distribution (equal probability on each configuration), π^{rule}: rule-based policy.
Output: θ: final RL policy parameters

function GENET(Ω, π^{rule})
  $\theta \leftarrow$ Random initial policy parameters
  $\Omega_{cur} \leftarrow \Omega$ ▷ Ω_{cur} will be updated and used for training
  for from 1 to N_{iter} **do** ▷ # of exploration iterations
    BO.INITIALIZE(Ω) ▷ Initialize with full config space Ω
    for from 1 to $N_{boTrials}$ **do** ▷ # of trial configs by BO
      $p \leftarrow$ BO.GETNEXTCHOICE()
      $adv \leftarrow$ CALCBASELINEGAP($p, \pi_\theta^{rl}, \pi^{rule}$)
      BO.UPDATE(p, adv)
    end for
    $p_{new} \leftarrow$BO.GETDECISION()
    ▷ Weight new config p_{new} by w and old configs by $1 - w$
    $\Omega_{cur} \leftarrow (1 - w) \cdot \Omega_{cur} + w \cdot \{p_{new}\}$
    $\theta \leftarrow$UNIFORMDOMAINRAND($\Omega_{cur}, \theta, N_{iters}$)
  end for
  return θ
end function
function CALCBASELINEGAP($p, \pi_\theta^{rl}, \pi^{rule}$)
  Initialize: $samples \leftarrow \emptyset$
  for 1 **to** N_{Tests} **do** ▷ # of reward comparisons
    $E \leftarrow S(p)$ ▷ Create a simulated env by p_i
    rollout $\phi^{rl} \sim \pi_\theta^{rl}(\cdot; E)$ ▷ Rollout RL π^{rl}
    rollout $\phi^{rule} \sim \pi^{rule}(\cdot; E)$ ▷ Rollout rule-based π^{rule}
    add $Reward(\phi^{rule}) - Reward(\phi^{rl})$ to $samples$
  end for
  return MEAN($samples$)
end function

A.4 Testbed setup

ABR: To test our model on a client-side system, we first leverage the testbed from Pensieve [8], which modifies dash.js (version 2.4) to support MPC, BBA, and RL-based ABR algorithms. We use the "Envivio- Dash3" video which format follows the Pensieve settings. In this emulation setup, the client video player is a Google Chrome browser (version 85) and the video server (Apache version 2.4.7) run on the same machine as the client. We use Mahimahi [113] to emulate the network environments from our pre-recorded FCC [100],

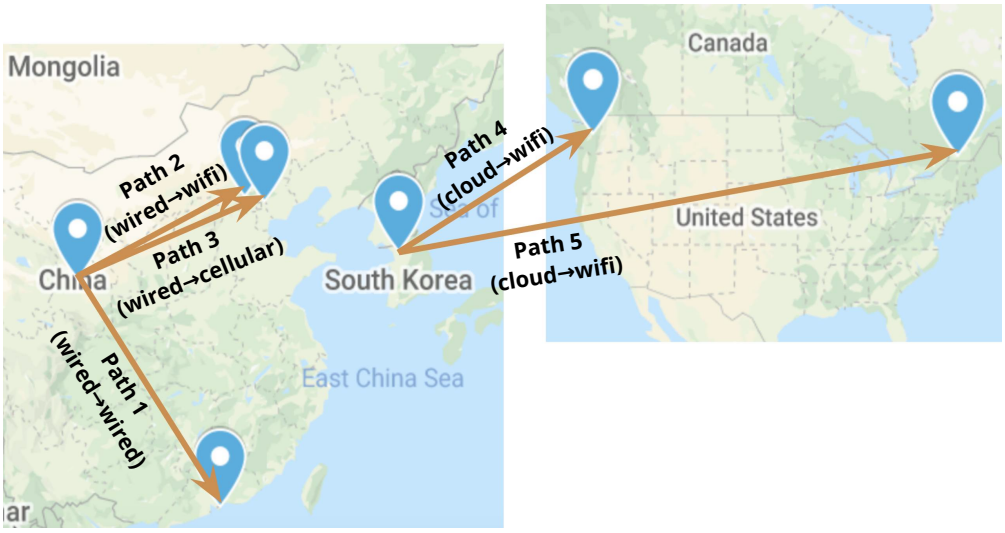


Figure A.1: Real-world network paths used to test ABR and CC policies.

cellular [122], Puffer [153] network traces, along with an 80 ms RTT, between the client and server. All above experiments are performed on UChicago servers.

CC: We build up CC testbed on Pantheon [151] platform on a Dell Inspiron 5521 machine. Pantheon uses network emulator Mahimahi [113] and a network tunnel which records packet status inside the network link. We run local customized network emulation in Mahimahi by providing a bandwidth trace and network configurations. We run remote network experiment by deplopying pantheon platform on the nodes shown in Figure A.1. Among all the CC algorithms tested, BBR [33] and TCP Cubic [60] are provided by Linux kernel and are called via iperf3. PCC-Aurora [71] and PCC-Vivace [49] are implemented on top of UDP. We train our models in python and Tensorflow framework and port the models into the Aurora C++ code.

Real network testbed: We also test the GENET-trained ABR and CC policies in real wide-area network paths (depicted in Figure A.1), including four nodes reserved from [6], one laptop at home, and two cloud servers.

	ABR	Bitrate (Mbps)	Rebuffering (s)	Bitrate change (Mbps)	Reward
Path 1	MPC	3.98	0.03	0.02	3.66
	BBA	3.84	0.018	0.15	3.51
	GENET	3.87	0.006	0.04	3.77
Path 2	MPC	3.22	0.041	0.07	2.74
	BBA	2.81	0.014	0.12	2.55
	GENET	3.15	0.008	0.07	3.01
Path 3	MPC	2.24	0.042	0.04	1.78
	BBA	1.75	0.03	0.05	1.40
	GENET	2.26	0.033	0.02	1.91
Path 4	MPC	2.93	0.013	0.04	2.76
	BBA	2.96	0.05	0.03	2.43
	GENET	2.88	0.002	0.02	2.84
Path 5	MPC	2.35	0.027	0.05	2.03
	BBA	1.82	0.022	0.04	1.56
	GENET	2.32	0.004	0.03	2.25

Table A.4: Reward breakdown of Figure 2.16(a) in ABR real-world experiment.

A.5 Details on reward definition

ABR: The reward function of ABR is a linear combination of bitrate, rebuffering time, and bitrate change. The bitrate is observed in kbps, and the rebuffering time is in seconds, and bitrate change is the bitrate change between bitrate of current video chunk and that of the previous video chunk. Therefore, a reward value can be computed for a video chunk. The total reward of a video is the sum of the rewards of all video chunks.

CC: The reward function of CC is a linear combination of the throughput (packets per second), average latency (s), and packet loss (percentage) over a network connection. In training, a reward value is computed using the above metrics observed within a monitor interval. The total reward is the sum of the rewards of all monitor intervals in a connection.

LB: The reward function of LB is the average runtime delay of a job set, which is measured by milliseconds. For each server, we observe its total work waiting time in the queue and the remaining work currently being processed. After the incoming job is assigned, the server would summarize and update the delay of all active jobs.

Path	CC	Throughput (Mbps)	90th percentile latency (ms)	Packet loss rate	Reward
Path 1	BBR	164.2	57.25	0.0906	-35.62
	Cubic	158.2	56.60	0.0072	104.2
	GENET	180.5	55.54	0.0063	152.1
Path 2	BBR	0.2108	3346	0.0407	-1721
	Cubic	0.2149	6978	0.2206	-4273
	GENET	0.1975	6381	0.0267	-3178
Path 3	BBR	5.40	1581	0.0136	-705.9
	Cubic	6.63	1400	0.0382	-719.1
	GENET	4.91	1180	0.0075	-439.9

Table A.5: Reward breakdown of Figure 2.16(b) in CC real-world experiments.

A.6 Baseline implementation

According to the paper [58], we train an additional RL model for Robustify to improve the main RL-policy model by generating adversarial network traces inside ABR. The state of the adversary model contains the bitrate chosen by the protocol for the previous chunk, the client buffer occupancy, the possible sizes of the next chunk, the number of remaining chunks, and the throughput and download time for the last downloaded video chunk. The action is to generate the next bandwidth in the networking trace, in order to optimize the gap between the ABR optimal policy, RL-policy, and the unsmoothness, which is the absolute difference between the last two chosen bandwidths. Here, the penalty of unsmoothness is set as 1, same as the paper.

We use PPO as the training algorithm, and train the Robustify adversary model with a RL model until they both converge. Afterward, we add the traces Robustify model generated into the RL training process to retrain the RL. The PPO parameter settings follow the original paper.

As an alternative implementation, we also use the reward defined in Robustify as the training signal for BO to search and update environments. For the unsmoothness penalty here, we empirically tried three numbers: 0.1, 0.5, 1. From our results, penalty=0.5 works better than others.

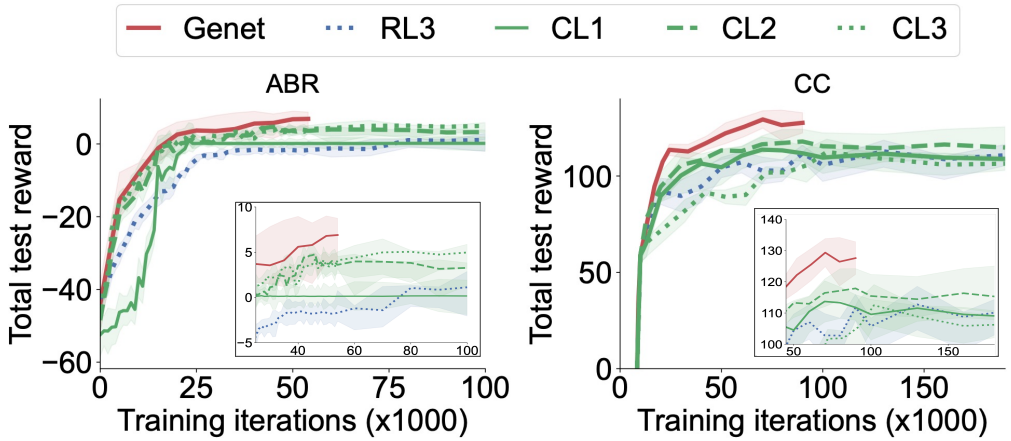


Figure A.2: Training RL and CL with more iterations still cannot outperform GENET.

A.7 Reward value breakdown

Table A.4 and Table A.5 contain the system metrics behind the reward values in Figure 2.16 for ABR and CC, respectively. The breakdown is done by decomposing the reward equations introduced in Table 2.1. For ABR, Table A.4 shows that GENET tends to train a model that leads to less rebuffering and more smoothed bitrate selection without significantly sacrificing the average bitrate. For CC, Table A.5 shows that GENET-trained model tends to have a lower 90th percentile latency and packet loss rate while not reducing throughput too much on Path 2 and 3. On Path 1, the performance gain is mainly from the larger throughput that GENET-trained model enables.

A.8 Train RLs and CLs with more iterations

To understand whether baselines like RLs and CLs can outperform GENET if they are given more training iterations, we trained RLs and CLs with twice as many training iterations as GENET. We empirically found that training with more iterations did not help the models trained by RLs and CLs as much as those trained by GENET. Their learning curves are shown in Figure A.2.