CausalSim: Toward a Causal Data-Driven Simulator for Network Protocols

Abdullah Alomar *  
MIT

Pouya Hamadanian *  
MIT

Arash Nasr-Esfahany *  
MIT

Mohammad Alizadeh  
MIT

Devavrat Shah  
MIT

Abstract

Evaluating the real-world performance of network protocols is challenging. Randomized control trials (RCT) are expensive and inaccessible to most researchers, while expert-designed simulators fail to capture complex behaviors in real networks. We present CausalSim, a data-driven simulator for network protocols that addresses this challenge. Learning network behavior from observational data is complicated due to the bias introduced by the protocols used during data collection. CausalSim uses traces from an initial RCT under a set of protocols to learn a causal network model, effectively removing the biases present in the data. Using this model, CausalSim can then simulate any protocol over the same traces (i.e., for counterfactual predictions). Key to CausalSim is the novel use of adversarial neural network training that exploits distributional invariances that are present due to the training data coming from an RCT. Our extensive evaluation of CausalSim on both real and synthetic datasets and two use cases, including more than nine months of real data from the Puffer video streaming system, shows that it provides accurate counterfactual predictions, reducing prediction error by 44% and 53% on average compared to expert-designed and standard supervised learning baselines.

1 Introduction

Causa Latet Vis Est Notissima — The cause is hidden, but the result is known. (Ovid: Metamorphoses IV, 287)

Evaluating the real-world performance of network protocols is challenging. The gold standard in the industry is to run randomized control trials (RCT). However, RCTs are time consuming, risk disruptions or SLA violations [30], and require significant infrastructure that is available only to large operators. The research community typically resorts to simulation or trace-driven experiments to assess new ideas. However, network simulators are notoriously unreliable in capturing real-world behavior [14], while trace-driven experiments (as we also show in this paper) are fraught with challenges due to data quality and bias issues [9,27,50].

This paper investigates the problem of learning a model to simulate the behavior of network protocols on real paths using offline historical data. Such a model, if accurate, would allow network operators to rapidly evaluate new ideas under real conditions without touching their networks. By contrast, even large operators with the infrastructure to run RCTs can currently evaluate only a small fraction of candidate ideas, due to the effort required to implement new proposals and the inherent risks of pushing new code to production networks. Moreover, if accurate models derived from real networks existed, operators could share them with the research community, enabling those without access to large networks to benchmark their ideas.

While obviously appealing, learning network behavior from offline data is difficult. As we discuss in depth in §2, the challenge lies in biases introduced into the observed data by the protocols used during data collection. Consider, for example, a trace of the execution of an adaptive bitrate (ABR) algorithm [24,36,45,50,51] during a video streaming session. Observed quantities, such as the throughput achieved when the player downloads a video chunk, are caused by certain latent properties of the network environment, such as the underlying network capacity and the particular choices made by the ABR algorithm (the bitrates chosen for each chunk). The observed data reflects the combined effect of these causes. But to simulate a new algorithm, we need to identify the underlying latent factors and the causal relationships that lead to the observations. Only then can we answer counterfactual questions needed for simulation, e.g., what would have happened had we applied a variant of the ABR algorithm on the observed trace with the same latent conditions?

We present CausalSim, a framework for building causal, data-driven models of network protocols. CausalSim can be applied in a wide range of networking problems, such as ABR algorithms, congestion control, load balancing, and more. CausalSim starts with traces collected using an initial RCT under a set of protocols. It uses this data to learn a causal model of the underlying network dynamics. CausalSim’s model can then be used to simulate any protocol (including new ones not part of the RCT) on any of the traces collected during the RCT. Therefore, while CausalSim requires an RCT for training, it greatly extends the utility of this initial RCT by enabling the evaluation of new unseen protocols.

CausalSim’s model simulates a new protocol in two steps. First, it infers the time-varying latent factors, corresponding to underlying network conditions, based on the observed data. Then, it predicts the evolution of the observed variables based on the learned latent factors and the actions taken by the protocol. This two-step process effectively removes the bias introduced by the protocol used in the observed trace, enabling accurate counterfactual predictions.

The key technical ingredient underpinning CausalSim is a

*Equal contribution
novel use of adversarial neural network training [17] that exploits a basic property of the training data; since it was collected via an RCT, we can expect the distribution of exogenous latent factors — such as the underlying capacity, which is an inherent property of a network path — to be policy invariant. In other words, these latent factors should have the same distribution across users assigned to different policies. Remarkably, the constraint imposed by this invariant is sufficient to infer latent factors with high predictive power.

We evaluate CausalSim on two use cases, ABR and server load balancing, with both real-world and synthetic datasets. Our main findings are:

1. Evaluation of CausalSim on more than nine months of real data from the Puffer [50] video streaming system shows 44% and 53% reduction in average counterfactual prediction error with respect to the expert-designed baseline and the standard supervised learning baseline, respectively.

2. Evaluation of CausalSim on a synthetic ABR dataset shows a Mean Absolute Percentage Error (MAPE) of only 6%.

3. Evaluation of CausalSim on a synthetic load balancing dataset shows that its internal representation has extremely high accuracy — Pearson Correlation Coefficient (PCC) of 0.99 with the true job size.

Our work has several limitations (§7). We still need to understand how CausalSim performs on other challenging problems, particularly those with high-dimensional latent states, and how the number and selection of policies observed in the training data affect the efficacy of our method. Nevertheless, we believe CausalSim provides an important first step towards building causal data-driven models of networked systems.

2 Motivation

Suppose a graduate student designs a new Adaptive Bit Rate (ABR) algorithm and calls it FabABR (Fabulous ABR). They claim that FabABR leads to 50% lower rebuffering events than BBA [24], according to simulations on a set of network traces. A video streaming company (e.g., Netflix) becomes interested in FabABR and wishes to evaluate it on its own network. The standard approach to do so would be to run a Randomized Control Trial (RCT). However, Netflix cannot afford to run an RCT for each new candidate algorithm. We pose two questions. (i) Is it possible to predict the performance of FabABR using historical observational data? (ii) Further, could Netflix use historical data to build a realistic data-driven model of their network, enabling researchers to accurately evaluate their novel algorithms without direct access to the real network?

2.1 Policy evaluation without an RCT

This section describes two strawman approaches to address the two tasks above using offline observational data. To make matters concrete, we evaluate these two approaches using more than nine months of data from Puffer, a recently deployed system for evaluating video streaming techniques [50].

Puffer collects data from a continual RCT switching between several ABR algorithms. In our period of interest, the tested algorithms include BBA [24], two versions of BOLA-BASIC [45], and two versions of an algorithm called Fugu developed by the Puffer authors. The dataset we use includes more than 62 million chunk downloads of more than 32 thousand streaming sessions, totaling four years of streamed videos. To evaluate a new ABR algorithm, we may be interested in various performance measurements, e.g., buffer occupancy, rebuffering rate, chunk download time, chosen chunk sizes, etc. For simplicity, we focus only on predicting the distribution of playback buffer occupancy. The buffer dynamics capture the core behavior of ABR algorithms and serves as a good proxy for other metrics of interest like rebuffering.

Recall that an ABR algorithm selects which bitrate to use among several possible options when downloading each chunk (e.g., 2 seconds) of video. The Puffer dataset1 contains logs of chosen chunk sizes, available chunk sizes, achieved chunk download throughputs, and playback buffer levels. We let one of the BOLA variants, BOLA-BASIC v1, play the role of the new FabABR algorithm that we wish to evaluate. So our task is: predict the distribution of buffer level for the users assigned to BOLA-BASIC v1 (henceforth called FabABR) in the Puffer dataset, using only the data from the other algorithms.

2.1.1 Simulation via Expert Modeling (ExpertSim)

As our first strawman, we build a simple trace-driven simulator (ExpertSim) using our knowledge of how an ABR system works. ExpertSim models the playback buffer dynamics for each step, where a step corresponds to one ABR decision and the download of a single video chunk. Let \( \hat{c} \) be the throughput achieved in step \( t \) (for the \( i \)-th chunk) of a particular video streaming session, which used, say, the BBA algorithm. To simulate FabABR for the same user, ExpertSim assumes that the user would achieve the same throughput \( \hat{c} \) in each step under FabABR. It then models how the playback buffer would change based on the bitrate (hence chunk size) selected by FabABR in each step. Specifically, if \( b_{t-1} \) is the buffer level at the start of this step and the chunk size chosen by FabABR is \( s_t \), then the buffer at the end of the step is \( b_t = \max(0, b_{t-1} - s_t/\hat{c}_t) + T \), where \( T \) is the chunk duration2. This assumption is indeed used in practice. For example, both FastMPC [51] and FESTIVE [28] assume that the observed throughput is independent of the chosen bitrate.

Figure 1a shows the true distribution of buffer level for BBA and FabABR (BOLA-BASIC v1) users in the Puffer dataset (the two dashed lines), as well as the distribution predicted by running FabABR on the traces collected from BBA users using ExpertSim (solid blue line). We see that the predictions are inaccurate: the buffer distribution generated by ExpertSim closely follows the buffer distribution of BBA users (the “source” policy from which the traces were obtained) and fails to match

---

1 We select logs, available on the puffer website, with a minimum length of 500 chunks (≈17 minutes) and average observed throughputs below 7 Mbps.

2 The complete buffer dynamic equation is slightly more complex, to handle cases with full buffers.
the buffer distribution of FabABR users (the “target” policy which we intended to simulate).

2.1.2 Simulation via Supervised Learning (SLSim)

Given the results above, the natural conclusion one will reach is that the simple model of buffer dynamics used in ExpertSim does not accurately capture the actual system dynamics. As a next attempt, we turn to a purely data-driven approach.

We use standard supervised learning to train a neural network that models the step-wise dynamics of the system. The input to this neural network is the buffer level at the start of a step, the achieved throughput for the step (which, like in ExpertSim, we obtain from the trace for a specific video streaming session), and the chosen chunk size for this step. This model learns to predict the buffer level at the end of each step through back-propagation on the training dataset. Importantly, we exclude the logs for FabABR from the training data.

Figure 1a compares the distribution of buffer levels predicted via this approach (SLSim) for FabABR. As with ExpertSim results, we use the traces collected from BBA users as the source policy. The results are similar to ExpertSim; once again, the predicted buffer distribution is closer to BBA users than FabABR.

2.1.3 What Went Wrong?

To understand the limitations of ExpertSim and SLSim, we plot the distribution of Minimum Round Trip Time (RTT) and the achieved per-chunk throughput for users assigned to BBA and FabABR in Figure 1b and Figure 1c, respectively. Since the Min RTT is an inherent property of a network path, we would expect its distribution to be the same for users assigned to different ABR policies. Figure 1b shows the distribution of Min RTT for users assigned to FabABR and BBA at two time scales. At the time-scale of one month (data collected in Jan. 2021), the two curves are notably different, indicating that one month of data is insufficient to see this invariance, i.e., the inherent randomness in the distribution is masking this expected invariance across policies. However, over the time scale of 9 months, the two distributions of Min RTT across both policies agree nearly perfectly.

However, such an invariance should not be expected for achieved throughput. For example, an ABR algorithm that tends to choose lower bitrates may achieve a lower throughput compared to one that chooses higher bitrates in the same network conditions. This can occur, for example, because the congestion control protocol takes some time to discover the available bandwidth (e.g., in slow start), and therefore cannot fully utilize the underlying network capacity for small chunk downloads. We can see this effect in the Puffer dataset. Figure 1c shows the distribution of the achieved per-chunk throughput for users assigned to BBA and FabABR. They are clearly different, and the difference does not vanish as we increase the time scale from one to nine months.

Indeed, the fundamental flaw in both ExpertSim and SLSim is that they view the achieved throughput observed in a trace as an inherent property of that network path, similar to the Min RTT. However, the throughput achieved when downloading a video chunk is impacted by the actions taken by the ABR policy. This bias due to the correlation between throughput and the chosen actions of a policy is exactly what needs to be carefully corrected for when trying to simulate the behavior of a different policy on the same network.

2.2 Causal Inference to the Rescue!

If the traces provided the true network capacity when each chunk was downloaded (rather than only the achieved throughput), our problem would be simple. First, we would learn the relationship between network capacity and achieved throughput for different ABR decisions in our data (e.g., as a function of the chunk size downloaded). Then, to simulate a new policy for a given trace, we would start with the network capacity at each step and predict
the achieved throughput in that step based on the bitrate (and chunk size) selected by the policy. This works because, like Min RTT, the underlying capacity is an inherent property of a network path and is not affected by the ABR policy.

Unfortunately, the true underlying network capacity is a latent quantity — we do not observe it, and cannot use it directly in our simulator. The key challenge then is to infer this latent quantity from observational data. Concretely, in our running example, we wish to estimate the true network capacity in each step of a trace, using observations such as chunk size, achieved throughput, etc.

This may seem like an impossible task but inferring such latent confounders and using them for counterfactual prediction is the core issue in the field of causal inference [40,41]. In this paper, we present CausalSim, a framework for data-driven simulation that is particularly well-suited to networking problems. As discussed in §3 and §4, CausalSim exploits (1) the properties of latent factors in a simple network model and (2) certain distributional invariances present in training data collected via an RCT. By taking advantage of this structure in the data generation process, CausalSim can infer the latent factors underlying its input data, and learn a causal model of network dynamics. As an illustration, Figure 1a shows the predicted buffer occupancy distribution when simulating FabABR on the traces of users assigned to BBA using CausalSim. As we can see, it matches the ground-truth distribution obtained for FabABR users very closely.

3 Modeling Networks with Causality

In this section, we introduce a natural model for the dynamics of a network protocol. Our goal will be to learn the underlying model of a system from observed trajectories, and subsequently to use this model to simulate the trajectory for a policy of interest. We shall argue that the model we introduce has a particular structure that exists in many network systems, which enables such simulation, i.e., makes causal counterfactual estimation feasible.

3.1 Models

The Markov Decision Process (MDP) is a generic abstraction to model the evolution of controlled dynamical systems. In a typical discrete-time MDP, time is denoted by an index $t \geq 0$. Let $s_t$ denote the state at time $t$, taking value in state space $S$. At each time $t$, the policy takes an action $a_t$ amongst a set of possible actions $\mathcal{A}$, causing a transition to the next state $s_{t+1} \in S$. The transition is determined through a kernel $T(\cdot|\cdot,\cdot)$, where $T(s'|s,a)$ denotes the likelihood that the next state is $s'$ when action $a$ is taken in state $s$. Importantly, we shall always assume the transition kernel is deterministic, i.e. applying the same action in the same state will always lead the MDP to the same next state. As network systems are reproducible, this assumption is valid in our context.

Going back to our ABR example, the MDP's state constitutes both information about the network path (e.g., the network capacity, Min RTT, etc.) and information about the user's video player (e.g., the amount of playback buffer, achieved throughputs of previous chunks, etc.) at any given time. The ABR policy’s actions (i.e., the bitrate selected for each chunk) lead to changes in the user's player state, and possibly also to the network path. Importantly, not all of the MDP state is observed by the policy. For example, the user’s player (which executes the ABR policy) does not observe the network capacity or other latent factors such as the number of competing flows sharing the same network path. This creates a POMDP, where the action $a_t$ depends on a partial observation $o_t$ derived from the state $s_t$, rather than $s_t$ directly. Figure 2a shows the causal structure for a general POMDP.

Exogenous latent state. Typically, network conditions change primarily due to exogenous factors, beyond the impact of a particular user’s policy or actions. For example, the available network capacity for a typical video streaming session is primarily determined by the overall conditions on that network path (e.g., the bottleneck link speed, number of competing flows, whether competing traffic is congestion controlled or not, etc.) rather than the specific actions of the ABR policy. On the other
hand, a user’s actions directly impact what that user observes, e.g., the achieved throughput as discussed in §2. Figure 2b shows the causal structure for a special type of POMDP, in which the latent state (denoted as \(u_t\)) is entirely exogenous and not impacted by the actions. Assuming that the latent state is exogenous greatly simplifies the task of inferring the latent state from observations. In particular, an immediate consequence is that we only need to consider \(o_{t-1}, o_t,\) and \(a_{t-1}\) to predict \(u_t\), that is:

\[
P(u_t|o_{t-1}, a_t, o_{t-2}, a_{t-2}, ...)=P(u_t|o_{t-1}, a_{t-1})
\]  

(1)

Proxy observations of latent state. As a further simplification, in some problems, the observed state includes a proxy of the latent state, which is sufficient to determine the effect of the latent state on the evolution of the other observed quantities. Figure 2c shows the structure of such POMDPs. The latent state \(u_t\) affects an observed proxy \(e_t\), which then determines the transition kernel for the observations \(o_t\) used by the policy. It is worth emphasizing two differences between \(e_t\) and \(u_t\). First, \(e_t\) is observed, whereas \(u_t\) is latent. Second, \(e_t\) is impacted by the actions, whereas \(u_t\) is exogenous. We note that \(o_t\) is the particular subset of observed variables which is used by the policy to choose actions; further recall that our goal is to accurately simulate only \(o_t\) for a counterfactual sequence of actions. In our running ABR example, \(e_t\) is the achieved throughput when downloading a chunk. Notice that the achieved throughput depends on the ABR action, but once it is revealed, it suffices to determine the evolution of other observed quantities like the playback buffer. The existence of such proxy observations further simplifies the task of inferring the latent state \(u_t\), which now depends only on \(e_t\) and \(a_{t-1}\):

\[
P(u_t|e_t, o_{t-1}, a_t, o_{t-2}, a_{t-2}, ...)=P(u_t|e_t, a_{t-1})
\]  

(2)

In the ABR case, for example, the above equation says that to infer the latent state (e.g., underlying network capacity), it suffices to consider the achieved throughput and the bitrate (or size) of a chunk. Other observations like the playback buffer size can be ignored.

3.2 Formalizing Network Simulation

Given the above network model, we now formalize the simulation problem. Specifically, we are given \(N\) trajectories (network traces), collected under \(M\) specific policies. Also, let the \(i^{th}\) trajectory be of length \(H_i\). Per the model described above, the \(i^{th}\) trajectory includes the following: (i) underlying latent states \(\{u_{i1}^{H_i}\}\); (ii) observed proxies \(\{e_{i1}^{H_i}\}\); (iii) observed states \(\{o_{i1}^{H_i}\}\); and (iv) the actions taken \(\{a_{i1}^{H_i}\}\).

The primary goal is to predict observed states under a different policy, not observed in such trajectories. Formally, given a sequence of counterfactual actions \(\{\tilde{a}_t\}_{i=1}^{H_i}\), starting with observation \(o_{i1}\) and under the same set of latent states \(\{u_{i1}^{H_i}\}\), estimate the counterfactual observed states \(\{\tilde{o}_{i1}^{H_i}\}\).

3.3 Simulation via Counterfactual Estimation

We shall see how the simulation problem we described is equivalent to learning a particular conditional distribution. Consider estimating \(\tilde{a}_t\) for \(t=2\). This corresponds to estimating its distribution, \(P(\tilde{a}_{t1}, o_{11}, u_{21})\). In particular, keeping \(o_{11}, u_{21}\) fixed, as we vary value of \(\tilde{a}_{t1}\), we wish to understand how this distribution varies.

However, latent states \(\{u_{t1}\}_{t=1}^{H_i}\) are not observed in any of the trajectories and we can only rely on observational data to estimate \(\tilde{a}_{t1}\). The transition from observed state \(a_{t1}\) and latent state \(u_{t2}\) to the next observed state depends on the taken action. While one instance of the transition \(P(o_{t2} \mid a_{t1}, u_{t2})\) is present in the data, the counterfactual instance of interest \(P(\tilde{o}_{t2} \mid \tilde{a}_{t1}, \tilde{u}_{t2})\) is not. Since the actions that were taken in the observational data were collected using a specific policy, the observational data only include one view of what could have happened. That is, observed trajectories are fundamentally biased.

To overcome this challenge, we utilize the abstraction of a Structural Causal Model (SCM) from the causal inference literature [40,41] to model the dynamics, which is consistent with Figure 2c: for any \(t\),

\[
a_{t1} = f_{\text{policy}}(o_{t1})
\]  

(3)

\[
e_t = f_{\text{emission}}(o_{t1}, u_{t1})
\]  

(4)

\[
o_t = f_{\text{system}}(o_{t1}, e_t, a_{t1})
\]  

(5)

Note that to be able to evaluate a particular test policy \(\tilde{f}_{\text{policy}}\), we need to learn \(f_{\text{emission}}\) and \(f_{\text{system}}\) and estimate latent states \(u_t\). However, since the counterfactual prediction problem is highly under-specified [11] as we further explain in §4.1, it is crucial to cleverly constrain the latent state estimation. This, in general, is a difficult problem as appropriate constraints require an expert understanding of the estimation task itself. However, our goal with CausalSim is to build data-driven simulators with minimal knowledge of the particular network we are simulating. Hence, rather than exploiting a particular structure of the network dynamics, we utilize the fact that our training data comes from an RCT. In particular, in the next section, we show how data collected from an RCT leads to certain distributional invariances across policies that help constrain the estimation of latent states \(u_t\).

4 Simulation Algorithm

In this section, we develop the simulation algorithm building upon the framework introduced in the prior section. First, we provide the key intuitions behind CausalSim, our proposed algorithm. Subsequently, we describe the method and its implementation in detail.

4.1 Intuition

In this section, we provide a high-level intuition about our algorithm design. Recall that to evaluate a particular test policy we would need to estimate \(f_{\text{system}}, f_{\text{emission}}\) and the latent states \(u_t\). While learning \(f_{\text{system}}\) can be simply achieved through supervised learning, deducing latent states \(u_t\) and learning \(f_{\text{emission}}\) is more complicated. One way to estimate the latent
states is by learning a mapping from \( a_{t-1}, e_t \) to \( \hat{u}_t \), denoted by \( \hat{F}_{\text{extraction}} \). That is:

\[
\hat{u}_t = \hat{F}_{\text{extraction}}(a_{t-1}, e_t)
\]  

(6)

Further, let \( \hat{F}_{\text{system}} \) and \( \hat{F}_{\text{emission}} \) denote our estimates of \( F_{\text{system}} \) and \( F_{\text{emission}} \) respectively.

If we had counterfactual data in our training data, i.e. we knew what would have happened if another choice was made, training \( \hat{F}_{\text{extraction}} \) would be simple. We would need to enforce that the extracted latent state is the same under different choices:

\[
\hat{F}_{\text{extraction}}(a_{t-1}, e_t) = \hat{F}_{\text{extraction}}(\hat{u}_{t-1}, \hat{e}_t)
\]  

(7)

In the absence of counterfactual data, we cannot impose such a constraint. Therefore, there are many possible estimates of \( \hat{F}_{\text{extraction}} \) and \( \hat{F}_{\text{emission}} \) with which we can fit the training data perfectly well (under-specification). For example, consider the case where the estimate of the latent state is simply the observed proxy \( e_t \). That is, we estimate \( \hat{F}_{\text{extraction}} \) to be the identity function \( \hat{F}_{\text{extraction}}(e_{t-1}, e_t) = e_t \). In this scenario, assuming that we successfully learn \( \hat{F}_{\text{system}} \), we can fit the training data perfectly. However, this function would likely violate Equation (7), since we know the proxy observation \( e_t \) is dependent on the observed action (see Figure 2c).

Therefore, choosing an estimate of \( u_t \) that achieves accurate predictions in the training data is insufficient and we need to impose constraints, achievable with only observational data.

To do so, CausalSim imposes an important distributional invariance. Specifically, recall that the available data we collect comes from running an RCT across a few policy choices. Given that actions do not affect latent states, and that the data is coming from an RCT, the distribution of latent states (i.e. \( u_t \)) should be invariant across policies. For example, as mentioned in §2.1.3, the distribution of minimum RTT is preserved across policies. Therefore, we add an additional constraint that the distributional characteristics of the estimated latent states should remain invariant across policies. This additional constraint reduces the size of the solution space for \( \hat{F}_{\text{extraction}} \). For example, it rules out the identity extraction function solution we presented earlier when used in the ABR example (see Figure 1c).

Remarkably, exploiting this constraint, CausalSim is able to infer latent factors with high predictive power, as we show in Section 5.

4.2 Detailed Implementation: CausalSim

Given the intuition above, we now provide a detailed description of the implementation of CausalSim. At a high level, CausalSim must learn to accurately predict observed states, while simultaneously ensuring it’s estimation of latent states remains policy invariant. CausalSim’s architecture and its three key components aim to achieve these goals, as depicted in Figure 3.

Each component is characterized by a neural network. In detail, these components are as follows:

- \( \hat{F}_{\text{extraction}} \), which corresponds to the map that utilizes \( a_{t-1}, e_t \) to estimate the latent states as in Equation (6). The neural network is represented by \( W_{\phi} \).
- \( \hat{F}_{\text{counterfactual}} \) which learns the combination of the two functions \( F_{\text{emission}} \) and \( F_{\text{system}} \):

\[
\hat{F}_{\text{counterfactual}}(\hat{u}_{t-1}, \hat{u}_t, \hat{a}_{t-1}) = \hat{F}_{\text{system}}(\hat{a}_{t-1}, \hat{F}_{\text{emission}}(a_{t-1}, \hat{a}_t), \hat{a}_{t-1})
\]  

(8)

The neural network is represented by \( W_{\psi} \).
- A Policy Discriminator, represented by \( W_{\gamma} \), that ensures policy invariance of the inferred latent states as discussed in §4.1. This guides \( \hat{F}_{\text{extraction}} \) in recovering the right latent states.

Algorithm 1 provides a detailed description of CausalSim and how it is implemented. We first go through achieving distributional invariance of latent states across policies, by connecting it to the recently emerging paradigm of adversarial learning in neural networks [18, 49]. Specifically, the policy discriminator \( W_{\gamma} \) aims to predict the policy \( \pi \) from the latent states \( \hat{F}_{\text{extraction}} (\mathcal{E}_0) \) generates, i.e., \( \hat{u} \). CausalSim uses a cross-entropy loss to train the policy discriminator:

\[
L_{\text{Disc}} = E_{D}[\log(W_{\gamma}(\pi[\hat{u}])]
\]  

(9)

where the expectation is over the training dataset denoted by \( D \). By Gibbs’s inequality [34], the optimal solution for this loss is:

\[
W_{\gamma_{\text{opt}}}(\pi[\hat{u}]) = P_{D}(\pi[\hat{u}])
\]  

(10)

Note that this solution depends on \( \hat{u} \) and thus on \( \hat{F}_{\text{extraction}} \), which is why next we consider the training of the black components in Figure 3. As stated in §4.1, the true latent states \( \pi_t \) are policy invariant because the data comes from an RCT. Therefore, if \( \hat{F}_{\text{extraction}} \) manages to infer the correct latent states, the optimal discriminator should not be able to detect the true policy. In other words, \( W_{\gamma_{\text{opt}}} \) would be a uniform distribution and the

\footnotetext{Training both \( \hat{F}_{\text{extraction}} \) and the discriminator in parallel causes instability. Therefore the discriminator should solely train (for num_disc it iterations) so that it converges and it’s loss curve plateaus.}
true latent states result in low discrimination accuracy. We use this as a signal during training to infer the true latent states, i.e.,
\[ L_{\text{fool}} = -L_{\text{Disc}} \] (11)

Further, we want the inferred latent states that enable \( L_{\text{counterfactual}} \) to accurately fit the observed states. This is the second signal we exploit during training, i.e.,
\[ L_{\text{pred}} = \mathbb{E}_D \left[ (o_t - P_\theta(o_{t-1}, a_{t-1}, \hat{u}_t))^2 \right] \] (12)

All in all, CausalSim combines both signals with a mixing hyper-parameter \( \kappa \) to train the black components in Figure 3:
\[ L_{\text{black}} = L_{\text{fool}} + \kappa \times L_{\text{pred}} \] (13)

5 Results

In this section, we do a systematic evaluation of CausalSim, using one real-world and two synthetic datasets. We compare CausalSim’s performance with two baselines:

• **ExpertSim** from §2.1.1: Uses a system expert’s knowledge for simulation.

• **SLSim** from §2.1.2: Uses a standard supervised-learning technique to learn dynamics of the system from the data.

5.1 Real-world ABR

Our real-world ABR dataset comes from Puffer [50], a research project aiming to evaluate ABR algorithms in an RCT setting with real video streaming sessions. Whenever a client initiates a video streaming session in Puffer’s website, a random ABR algorithm is chosen and assigned to that session. Sessions are logged (buffer levels, chunk sizes, timestamps, download times, etc.) anonymously and the data is available for public use. We use the trajectories in these logs as data for training and evaluation. These trajectories use 5 ABR algorithms (linear BBA, BOLA-BASIC v1, BOLA-BASIC v2, Fugu-CL, Fugu-2019), explained in more detail in §A.1.

5.1.1 Can CausalSim Simulate a New Policy?

To answer this question, we remove all trajectories that use one of the ABR algorithms from the dataset, train CausalSim on the rest and then simulate the left-out ABR algorithm. For this purpose, we can simulate BBA, BOLA-BASIC v1 and BOLA-BASIC v2, and choose one of them to be the left-out policy. We do not simulate Fugu, since it requires access to some TCP information which is not logged in the dataset (see §A.1). We follow Algorithm 1 to train CausalSim’s components \((L_{\text{counterfactual}}, \text{Policy Discriminator})\). The training setup is explained in detail in §A.2. Next, we use the trained CausalSim to simulate each choice of the left-out ABR algorithms, using the trajectories from the other ABR algorithms in the dataset, which we call the source.

Figure 4 shows the buffer level distributions of source users, left-out users, CausalSim’s simulations, ExpertSim’s simulations, and SLsim’s simulations, for 3 of the 12 possible combinations of source and left-out ABR algorithms. Ideally, a simulator should predict the ground-truth buffer distribution for the left-out users based on the data for any group of source users (since the RCT assigns users to policies at random). These three examples shown in the figure correspond to the scenarios where CausalSim has the best, median, and worst accuracy in terms of matching the ground-truth distribution of the left-out policy (see definition of metrics below). All 12 figures for different combinations of source/left-out users are shown in Figure 11 (§A.3).

To measure the performance of CausalSim in a more quantitative manner, we use the 1-Wasserstein distance, or the Earth Mover Distance (EMD) between two distributions:
\[ \text{EMD}(p, q) = \int_{-\infty}^{+\infty} |P(x) - Q(x)|dx \] (14)

where \( P \) and \( Q \) are the Cumulative Distribution Function (CDF)s of \( p \) and \( q \), respectively. The smaller the EMD between simulation and left-out distributions, the closer the distributions, and hence the better the performance of the simulator.

Figure 5a shows the CDF of EMD between the predicted and left-out distributions, over all possible source/left-out ABR algorithm pairs. We see that the EMD of CausalSim is smaller than the EMD of the baselines across almost all experiments. In about 30% of cases, SLSim is slightly better than CausalSim, but these are cases with low EMD (< 0.2), where all schemes perform quite well. Figure 11 shows the EMD values for CausalSim for all 12 source/left-out policy combinations.
Figure 4: Buffer level distribution of CausalSim’s simulation, baselines, the left-out ABR algorithm, and the source ABR algorithm, of a source left-out pair for which the relative performance of CausalSim with respect to the baselines is (a) perfect, (b) medium, (c) bad, in terms of the visual distributional closeness to the left-out buffer level distribution. The closer a scheme’s buffer level distribution to the left-out buffer level distribution, the better.

Figure 5: CausalSim has smallest Earth mover distance (EMD) from ground-truth. (a) Distribution of CausalSim, ExpertSim, and SLSim EMDs over all possible source left-out pairs. (b) Validation EMD and test EMD are highly correlated. This justifies our hyper-parameter tuning strategy. (c) Comparing the distribution of CausalSim EMDs in the whole population with smaller sub-populations.

5.1.2 How to Tune CausalSim’s Hyper-parameters?

Counterfactual prediction is not a standard supervised learning task that optimizes in-distribution generalization. Rather, it is always an Out of Distribution (OOD) generalization problem, i.e., we collect data from a training policy (distribution 1), and want to accurately simulate data under a different policy (distribution 2). Since we of course do not get data from the test policy when we train CausalSim, we use the following natural proxy for tuning hyper-parameters: Simulating ABR algorithms in the training data using trajectories of other ABR algorithms in the training data. This of course can be viewed as an OOD problem as well. We claim that if a choice of hyper-parameters results in a robust model that performs well OOD across all validation ABR algorithms in the training data, it should work well for the actual left-out test policy as well.

We verify this hyper-parameter tuning procedure empirically. For each choice of the three left-out ABR algorithms (hence training dataset), we train six different CausalSim models with different choices of \( \kappa \) (defined in §4.2). We consider two metrics: (i) Test EMD, defined as the average EMD when simulating the left-out ABR algorithm from the ABR algorithms in the training dataset. This is our main performance objective. (ii) Valid EMD, defined as the average EMD when simulating ABR algorithms in the training dataset from other ABR algorithms in the training dataset. This is our proxy objective for hyper-parameter tuning.

For each model (18 in all: 3 datasets, 6 hyper-parameters), we calculate both Test EMD and Valid EMD, which results in one \((\text{Valid EMD}, \text{Test EMD})\) point in Figure 5b. The PCC between Valid EMD and Test EMD is 0.94, which shows high linear correlation. Hence, though CausalSim might not always perform well (i.e., test EMD is not low for some combinations of training dataset and hyper-parameters), we can have a very good idea of how well it works by measuring Valid EMD.

5.1.3 Can we do a More Fine-grained Evaluation?

Ideally, we would like to evaluate CausalSim’s counterfactual predictions on a step-by-step basis for a given trajectory. However, this is not possible in real-world data, as we only see the outcome of one ABR algorithm’s chosen action for a single step. In other words, there is no way to get ground truth for individual counterfactual predictions in the observational data, which is
referred to as the fundamental problem of Causal Inference [21]. This is the reason we evaluated predictions on a distributional level, rather than on a step-by-step basis in Figure 5a, using the fact that the RCT gives us ground truth for distributions of quantities of interest (like the buffer occupancy) for each policy.

However, there is a way to evaluate CausalSim’s predictions at a more fine-grained level. Instead of evaluating the predicted distribution of buffer occupancy across the whole population, we can evaluate on certain sub-populations of users. The only requirement is that the way we select these sub-populations should be statistically independent of the ABR algorithm. For example, we can partition users by a metric such as Min RTT, which is independent of the policy chosen for each user in the RCT (see discussion of Figure 1b in §2.1.3).

We use the MinRTT to create the following four sub-populations:
1. users with Min RTT < 35\(\text{ms}\)
2. users with 35\(\text{ms}\) ≤ Min RTT < 70\(\text{ms}\)
3. users with 70\(\text{ms}\) ≤ Min RTT < 100\(\text{ms}\)
4. users with 100\(\text{ms}\) ≤ Min RTT

Now, we can ask question of the following type: had the users in sub-population two, who were assigned the source ABR algorithm, instead used the left-out ABR algorithm, what would the distribution of their buffer level look like? As the ground truth answer to this question, we can use the buffer level distribution of users in sub-population two assigned to the left-out policy.

Figure 5c shows the CDF of CausalSim’s EMD when simulating the left-out ABR algorithm over the whole population, and each of the above sub-populations. We can see that the EMD achieved for sub-populations is similar to the whole population, indicating that CausalSim’s predictions are equally good across the sub-populations.

5.2 Synthetic ABR

We now evaluate CausalSim in a synthetic ABR environment, in which we can obtain ground truth for individual counterfactual predictions on a step-by-step basis for a trajectory. In these experiments, we also use a larger set of policies than available in the real data.

5.2.1 Simulation Dynamics

In each simulated training session, we start with an empty playback buffer and a latent network path characterized by an RTT and a capacity trace. In each step, an ABR algorithm chooses a chunk size, which is transported over this network path to the client as the buffer is depleting. Once the user receives the chunk, the buffer level increases by the chunk duration. This simple system can be modeled as follows:

\[
b_t = \min(b_{t-1} - d_t, 0) + c
\] (15)

where \(b_t\), \(d_t\) and \(c\) refer to the buffer level at time step \(t\), the download time of the chunk at time step \(t\), and the chunk video length in seconds, respectively. Streaming the next chunk is started immediately following receiving the previous one, except when the buffer level surpasses a certain value (in our case, 20 seconds). To compute \(d_t\), we model the transport as a Transmission Control Protocol (TCP) session with an Additive Increase - Multiplicative Decrease (AIMD) congestion control mechanism with slow start. For every chunk, the TCP connection starts from the minimum window size of 2 packets and increases the window according to slow start. Therefore, it takes the transport some time to begin fully utilizing the available network capacity. The overhead incurred by slow start depends on the RTT and bandwidth-delay product of the path. When downloading chunks with large sizes, the probing overhead is minimal but it can be significant for small chunks. Therefore, as we observed in the Puffer data, the throughput achieved for a given chunk in this synthetic simulation depends on the size of the chunk.

Our simulation consists of 5000 trajectories, for each of whom a policy is randomly chosen from the list of available policies. For a full explanation of the simulation setup, network capacity generation procedure, and the simulated policies, refer to §B.1.

Performance Metric: We compare CausalSim predictions with ground truth counterfactual trajectories, via the MSE distance.
5.2.2 Can CausalSim Faithfully Simulate New Policies?

Similar to our real-data evaluations, we train models based on training data generated using all policies except a left-out policy, for which the model does not observe any data. Although traces come from the same generative process, no two trajectories in the dataset collected with different policies share the exact same trace, as this would be an unrealistic data collection scenario. Given that we have 9 possible policies to leave out, we have 9 possible datasets and models. There are 8 possible groups of trajectories to choose as sources, based on the policy that generated them. In total this leaves 72 different combinations and scenarios. We use the same hyper-parameter tuning approach examined in §5.1.2. Figure 6a compares the CDF of MSE values resulting from CausalSim and the two baselines. As evident, both baselines suffer from inaccurate predictions and in some cases are catastrophically inaccurate. On the contrary, CausalSim maintains favorable performance, even in the tail of its MSE distribution. Figure 6b gives a closer look at the CDF curves below their median. We see CausalSim dominates at every scale.

Figure 6c is a heatmap of the two dimensional histogram of CausalSim predictions and ground truths. A fully accurate prediction scheme would perfectly match the ground truth and only the diagonal of this histogram would be populated. CausalSim almost achieves that, indicating it produces accurate trajectories on a step-by-step basis.

Further, in Figure 7, we compare the the Mean Absolute Percentage Error (MAPE) of CausalSim, ExpertSim and SLSim predictions across all trajectories at each time step for the first 35 steps. Note that the error naturally accumulates for all three methods as we move forward in time. However, CausalSim maintains a MAPE of (~6%) which significantly lower than ExpertSim’s (~15%) and SLSim’s (~9%).

\[
MSE(p, q) = \|p - q\|_2^2
\]  

(16)

Here, \(p = \{p_t\}_{t=1}^N\) and \(q = \{q_t\}_{t=1}^N\) are time series vectors. Better predictions yield smaller MSE values, where an ideal MSE is 0.

5.3 Synthetic Load Balancing

Thus far, our examples and evaluations have centered around ABR. However, we now show that CausalSim is a more general framework for answering counterfactual queries that are of interest in networking.

5.3.1 Environment Setup and Simulation Dynamics

We consider a Load Balancing problem with heterogeneous servers, as depicted in Figure 8. This environment consists of \(N = 8\) servers (and a queue for each) with different processing powers, a scheduler, and a series of jobs that need to be processed on these servers. Each job has a specific size which is unknown to the scheduler as well. Each server can process jobs at a specific rate \(r_{ik}\), which is unknown to the scheduler as well. The scheduler will receive jobs and assign them to one of \(N\) servers. Assuming the \(k^{th}\) arriving job has size \(S_k\) and gets assigned to server \(a_k\), the job processing time will be \(S_k \cdot r_{ak}\).

If this job is not blocked by some other job being processed, it will incur a latency equal to its processing time. If it is blocked and the jobs ahead of it in the queue take \(T_k\) to be processed, the incurred latency is \(S_k \cdot r_{ak} + T_k\). The goal is to design a server assignment algorithm for the scheduler that minimizes job completion latencies. At each point, the algorithm can observe the lengths of several or all server queues.

In our simulation, we generate a collection of 5000 trajectories each with 1000 steps and use 16 policies in the scheduler. For a detailed explanation of the policies and job size and server processing rate generation processes, refer to §C.1.

5.3.2 Experiment setup

The aim of this experiment is to evaluate whether we can simulate new unseen test policies in this environment, given observations of (processing time and latency) from other training policies. Note that while we observe the processing time and the latency of each job, the actual size of the job is not observed, i.e., it is the latent states in this problem.

In the absence of an expert baseline analogous to that of the ABR experiments, we compare CausalSim against SLSim. Specifically, in this adapted variant of SLSim, the input to the neu-
As is done in the ABR case studies, we train CausalSim and 5.3.3 Can CausalSim Faithfully Simulate New Policies?

predicting for each pair of source-target policies, e.g., counterfactual for the tracker policy for trajectories generated by the shortest queue policy. In total, we have 120 different source/left-out policy pairs across the eight datasets.

In Figure 9a and Figure 9b, we show the CDF of the MSE of predicting the processing time and the latency, respectively, using both CausalSim and SLSim. As evident in these two figures, CausalSim error is significantly lower than that of SLSim for both the processing time and latency. Further evidence of the accuracy of CausalSim counterfactual predictions is illustrated in Figure 9c and Figure 9d. These two figures show that the counterfactual point-wise predictions (i.e., for each individual job) are very similar to the ground truth values, as evident by the highly populated diagonal line in both plots.

5.3.4 Does CausalSim Faithfully Infer Latent States?

We test the claim that the architecture used in CausalSim of estimating the exogenous latent state and using it to predict the next state is key to producing accurate counterfactual predictions. To do so, we compare CausalSim’s estimated latent state with the underlying job sizes—the job size is indeed the latent state that dictates the dynamics in the load balancing environment.

As desired, we find that the estimated latent states and the job sizes are highly correlated, as illustrated in Figure 10, with a PCC of -0.991. This demonstrates that CausalSim can learn faithful representations of true latent states, as was the goal behind its architecture.

6 Related-Work

Data-driven simulation and policy evaluation in systems and networking: Packet level simulators [10, 20, 31] tend to sacrifice either scalability or granularity when simulating large networks, and can lack realism in configurations. Recent research combines Machine-Learning with packet-level modeling to improve speed and accuracy of data-center [53] and network [8] simulators. There has been a surge in the use of data-driven optimization in network systems [35–37] to propose superior decision algorithms. These works often evaluate their proposals on a set of traces, but trace-driven evaluation can be problematic [9, 50] because of the

Figure 10: Two-dimensional histogram heatmap of CausalSim extracted latent state vs. latent job sizes, when the random policy is the left out test policy. Similar results are obtained with other policies as well.
skews in the traces caused by the logging strategy (e.g. the policy used while collecting data) and also lack of coverage for certain sub-populations. Re-weighting schemes such as like Inverse Propensity Scoring [22] aim to mitigate the aforementioned skew, by accounting for the difference between the logging and the evaluation policy [26,32]. Re-weighting schemes suffer from variance, especially in long trajectories, if the logging and evaluation policies are not similar. Furthermore, these schemes rely on unrealistic stochastic collection policies, such as random policies. Doubly Robust techniques [9] aim to reduce both variance and bias by combining model-based and re-weighting based methods. Generally, policy evaluation methods cannot respond to counterfactual questions at sub population or personal levels, such as a single trajectory. What-If Scenario Evaluator (WISE) [48] builds a Causal Bayesian Network from the data that is able to answer what-if (intervention) questions. However, their method requires no latent confounding variables in the problem setting.

Unlike methods of policy evaluation, CausalSim can do counterfactual inference, with which one can do policy evaluation as well. The closest work to ours [46] considers the problem of counterfactual inference for ABR. However, their method needs a model of the underlying network capacity, which is not available in real-world scenarios.

**Robust Representation Learning:** Domain adaptation is a problem of learning in a source domain for a good performance in a target domain, from which you only have unlabeled data. Adversarial Discriminative Domain Adaptation (ADDA) [49] uses a training approach similar to ours, in order to discover features with the same distribution in the source and target domains, which they claim to be robust features. However, we introduce the invariance in the training data collection ourselves and further exploit it for learning the right models. Furthermore, there has been recent interest in learning causal representations for improving robustness and generalization capability of neural networks [44]. CausalSim learns the causal representations using its novel data collection and its adversarial training procedure.

**Causal Inference:** Identifying causal relationships from observational data is a critical problem in many domains [19], including medicine [38], epidemiology [42], economics [25], and education [13]. Indeed, identifying causal structure and answering causal inference queries is an emerging theme in different machine learning tasks recently, including computer vision [52], reinforcement learning [15], and fairness [16], to name a few. One important aspect about causal inference is its ability to answer counterfactual queries. That is, answering queries about whether an outcome would have occurred had some intervention taken place. For such queries, many methods were developed; where some approaches are motivated by Pearl’s structural causal model [40], and by Rubin’s potential outcome framework [43]. We refer the interested reader to recent surveys such as [19] and references there in for an overview of recent advances in our ability to infer causal relationships from observational data.

The most closely related line of work within this literature we build upon is synthetic controls and its extension synthetic interventions, which aims to build synthetic trajectories of different units (e.g. individuals, geographic locations) under unseen interventions by appropriately learning across observed trajectories [1, 2, 4–7]. However, these works do not directly apply to network problems or to settings with adaptive policies.

A recent work which does consider such policies is [3]. This work aims to build data-driven simulators for model-based offline reinforcement learning RL, where we get limited data from different agents. The authors utilize a low-rank factor model approach to do so. However, their evaluations are primarily focused on more standard RL benchmarks from the OpenAI gym. Possibly utilizing a low-rank factor model approach with the distributional invariance method used in this paper might lead to further data efficiency and increased simulation accuracy. We leave this as interesting future work.

## 7 Concluding Remarks

CausalSim provides a simple and general solution for evaluating the performance of network protocols, as we showcase in this paper through two use-cases using both synthetic and real-world data. The success of this approach has many implications. One major implication is that RCT data can be harnessed to provide a more robust and accessible benchmark for researchers to evaluate new algorithms. Further, our evaluations with synthetic data show that CausalSim provides accurate point-wise counterfactual predictions; this suggests that one may be able to use CausalSim to design and evaluate personalized policies for specific scenarios/users.

While successful, we believe that a few more steps are needed for CausalSim to be a standard solution to counterfactual/causal inference problems in networked systems. One immediate next step is to go beyond ABR and load balancing and evaluate CausalSim’s performance in other types of systems. For example, while CausalSim is proven successful in problems with low-dimensional states, it is yet unclear if that success will extend to problems with a high-dimensional latent state. Another important aspect that needs further exploration is to understand how data availability affects the efficacy of this approach. For example, we suspect that the efficacy of CausalSim increases as the number and diversity of policies observed in the training data increase; however, a systematic evaluation of such a claim is needed and is an interesting direction for future work. Indeed, it would be worthwhile knowing which type of policies we should collect RCT data on, such that it allows us to best simulate new test policies.

Finally, a limitation of our approach is that it requires the availability of RCT data, which is not always available. However, RCT data is becoming more prevalent, and many network operators and service providers already have some infrastructure for conducting RCT experiments. Also, as CausalSim promises to increase the utility of such trials, performing RCTs may become more appealing. However, proposing a solution that overcomes the RCT data requirement is another interesting direction of future work.
References

[1] A. Abadie, A. Diamond, and J. Hainmueller. Synthetic control methods for comparative case studies: Estimating the effect of california’s tobacco control program. *Journal of the American Statistical Association*, 2010.

[2] A. Abadie and J. Gardeazabal. The economic costs of conflict: A case study of the basque country. *American Economic Review*, 2003.

[3] Anish Agarwal, Abdullah Alomar, Varkey Alumootil, Devavrat Shah, Dennis Shen, Zhi Xu, and Cindy Yang. Persim: Data-efficient offline reinforcement learning with heterogeneous agents via personalized simulators. *arXiv preprint arXiv:2102.06961*, 2021.

[4] Anish Agarwal, Devavrat Shah, and Dennis Shen. Synthetic interventions. *arXiv preprint arXiv:2006.07691*, 2021.

[5] Anish Agarwal, Devavrat Shah, Dennis Shen, and Dogyoon Song. On robustness of principal component regression. *Journal of the American Statistical Association*, 2021.

[6] Muhammad Amjad, Vishal Misra, Devavrat Shah, and Dennis Shen. Mrsc: Multi-dimensional robust synthetic control. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(2), June 2019.

[7] Muhammad Amjad, Devavrat Shah, and Dennis Shen. Robust synthetic control. *Journal of Machine Learning Research*, 19(22):1–51, 2018.

[8] Sachin Ashok, Sai Surya Duvvuri, Nagarajan Natarajan, Venkata N Padmanabhan, Sundararajan Sellamanickam, and Johannes Gehrke. ibox: Internet in a box. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 23–29, 2020.

[9] 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.

[10] Xinjie Chang. Network simulations with opnet. In *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 1,WSC ’99*, page 307–314, New York, NY, USA, 1999. Association for Computing Machinery.

[11] Alexander D’Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 2020.

[12] DASH Industry Form. Reference client 2.4.0, 2016.

[13] Rajeev H Dehejia and Sadek Wahba. Causal effects in nonexperimental studies: Reevaluating the evaluation of training programs. *Journal of the American statistical Association*, 94(448):1053–1062, 1999.

[14] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACm Transactions on Networking*, 9(4):392–403, 2001.

[15] Andrew Forney, Judea Pearl, and Elias Bareinboim. Counterfactual data-fusion for online reinforcement learners. In *International Conference on Machine Learning*, pages 1156–1164. PMLR, 2017.

[16] Sahaj Garg, Vincent Perot, Nicole Limtiaco, Ankur Taly, Ed H Chi, and Alex Beutel. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 219–226, 2019.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[19] Ruocheng Guo, Lu Cheng, Jundong Li, P Richard Hahn, and Huan Liu. A survey of learning causality with data: Problems and methods. *ACM Computing Surveys (CSUR)*, 53(4):1–37, 2020.

[20] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[21] Paul W Holland. Statistics and causal inference. *Journal of the American statistical Association*, 81(396):945–960, 1986.

[22] Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

[23] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 internet measurement conference*, pages 225–238, 2012.

[24] 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
[25] Guido W Imbens. Nonparametric estimation of average treatment effects under exogeneity: A review. *Review of Economics and statistics*, 86(1):4–29, 2004.

[26] 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.

[27] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. Unleashing the potential of data-driven networking. In *International Conference on Communication Systems and Networks*, pages 110–126. Springer, 2017.

[28] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108, 2012.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] S Shummuga Krishnan and Ramesh K Sitaraman. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[31] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[32] 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.

[33] Dong Lu, Yi Qiao, P.A. Dinda, and F.E. Bustamante. Characterizing and predicting tcp throughput on the wide area network. In *25th {IEEE} International Conference on Distributed Computing Systems (ICDCS’05)*, pages 414–424, 2005.

[34] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[35] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets ’16, page 50–56, New York, NY, USA, 2016. Association for Computing Machinery.

[36] 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.

[37] Hongzi Mao, Malte Schwarzkopf, Shaileshh Boja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM ’19, page 270–288, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Cross-Disorder Group of the Psychiatric Genomics Consortium et al. Identification of risk loci with shared effects on five major psychiatric disorders: a genome-wide analysis. *The Lancet*, 381(9875):1371–1379, 2013.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[40] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009.

[41] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[42] James M Robins, Miguel Angel Hernan, and Babette Brumback. Marginal structural models and causal inference in epidemiology, 2000.

[43] Donald B Rubin. Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331, 2005.

[44] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

[45] Kevin Spiteri, Rahul Uragonkar, and Ramesh K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020.
[46] Panchapakesan C Sruthi, Sanjay Rao, and Bruno Ribeiro. Pitfalls of data-driven networking: A case study of latent causal confounders in video streaming. In *Proceedings of the Workshop on Network Meets AI & ML*, pages 42–47, 2020.

[47] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM ’16, page 272–285, New York, NY, USA, 2016. Association for Computing Machinery.

[48] Mukarram Tariq, Amgad Zeitoun, Vytautas Valancius, Nick Feamster, and Mostafa Ammar. Answering what-if deployment and configuration questions with wise. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 99–110, 2008.

[49] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[50] 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.

[51] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. *SIGCOMM Comput. Commun. Rev.*, 45(4):325–338, August 2015.

[52] Dong Zhang, Hanwang Zhang, Jinhui Tang, Xian-sheng Hua, and Qianru Sun. Causal intervention for weakly-supervised semantic segmentation. *arXiv preprint arXiv:2009.12547*, 2020.

[53] Qizhen Zhang, Kelvin K. W. Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. Mimicnet: Fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM ’21, page 287–304, New York, NY, USA, 2021. Association for Computing Machinery.
Appendix A  Real-world ABR
A.1 Dataset & Algorithms
Our trajectories in the real-world (Puffer) data come from the time span of August 1, 2020 until May 15, 2021. In this period of time, 5 ABR algorithms appear consistently and are listed in Table 1. Each trajectory is an active client session streaming a live TV channel. We used trajectories with a minimum length of 500 chunks to filter out short-lived sessions. The observed throughputs are in the range of 200Kbps to 300Mbps. Since the highest quality chunks rarely surpass 7Mbps, paths with higher bandwidth will always stream the highest quality chunks under all policies. Hence we only use trajectories with less than 7Mbps of average observed bandwidth.

To test out CausalSim, we need to simulate the streaming session using a different algorithm than the one that was actually used in that session. This requires implementation of the ABR algorithms. However, Fugu’s implementation needs access to the history of the streaming session’s TCP info, which is not fully logged in the data-set. More specifically, our attempts for reconstructing Fugu’s chosen chunk sizes fail to match the logged ones in the data-set. Therefore, we do not consider Fugu-2019 or Fugu-CL as candidates for left-out algorithms.

A.2 Training setup
We use Multi Layer Perceptron (MLP)s as the neural network structures for CausalSim models and the SLSim model. All implementations use the Pytorch [39] library. Table 2 is a comprehensive list of all hyperparameters used in training.

A.3 Comprehensive results
Figure 11 compares CausalSim predictions with ExpertSim and SLSim baselines in all possible source/left-out policy pairs.

Appendix B  ABR synthetic
B.1 Data & Algorithms
Simulating a trajectory in our synthetic ABR environment needs three components:
• A video, with several bit-rates available. We use "Envivio-Dash3" from the DASH-246 JavaScript reference client [12].
• An ABR algorithm. We have a set of 9 policies to choose from, presented in Table 3.
• A network path, which is characterized by the latent network capacity and the path RTT.

We use random generative processes to generate network traces and RTTs. The RTT for a streaming session is sampled randomly, according to a uniform distribution:

\[ rtt \sim \text{Unif}(20 \text{ ms}, 200 \text{ ms}) \]

Our trace generator is a bounded Gaussian distribution, whose mean comes from a Markov chain. Prior work shows Markov chains are appropriate models for TCP throughput [47], and Gaussian distributions can model throughputs in stationary segments of TCP flows [33].

Concretely, at the start of the trace, the following parameters are randomly sampled:

\[ v \sim \text{Unif}(30, 100) \]
\[ p = 1/v \]
\[ l, h \sim \text{Unif}(0.5, 4.5) \]
\[ s.t. \frac{h-l}{h+l} > 0.3 \]
\[ s_0 \sim \text{Unif}(l, h) \]
\[ c_\sigma \sim \text{Unif}(0.05, 0.3) \]

At each time step, the state remains unchanged with probability \(1-p\) and changes elsewhere. When changing, the next state is sampled from a double exponential distribution centered around the previous state:

\[ \lambda = \text{solve}(1 - e^{h-s_{i-1}} - e^{h-s_{i-1}} = 0) \]
\[ s_i = \text{DoubleExp}(s_{i-1}, \lambda) \]

The point for this specific transition kernel is, that abrupt changes in network capacity should be rare. The scale parameter in this distribution Finally, the network capacity \(c_i\) in each step is sampled from a Gaussian distribution, defined by these parameters:

\[ c_i \sim \text{Normal}(s_i, s_i \cdot c_\sigma) \]

B.2 Training setup
Similar to the real-world ABR experiment, we use MLPs as the neural network structures for CausalSim models and the SLSim model. Table 4 comprehensively lists all hyperparameters used in training.

Appendix C  Load Balancing
C.1 Data & Algorithms
To simulate the load balancing problem described in §5.3.1, we need to set the server processing rates \(\{r_i\}_{i=1}^N\), and arriving job sizes \(S_i\). Server rates are generated randomly, as follows:

\[ r_i = e^{u_i} \]

where \(u_i \sim \text{Unif}(-\ln(5), \ln(5))\)

We generate job sizes using a time-varying Gaussian distribution. At step \(k\) of the trajectory, job size \(S_k\) is sampled as follows:

\[ S_k \sim \text{Normal}(\mu_k, \sigma_k) \]

where \(\mu_k\) and \(\sigma_k\) signify the mean and variance of the generative distribution at time step \(k\). At each time step, with a probability of \(p = 1/12000\), the mean and variance change and with a probability of \(1-p\), they remain the same. The mean and variance...
Figure 11: Buffer distributions for Source users, Left-out users, CausalSim predictions, ExpertSim predictions, SLSim predictions.
values are drawn from random distributions, both at the start of a trajectory and when a change occurs, in the following manner:

If $k=0$ or, mean and variance must change:
\[
\begin{align*}
\mu_k &\sim \text{Pareto}(\alpha = 1, \ L = 10^1, \ H = 10^{2.5}) \\
\rho_k &\sim \text{Unif}(0, \ 0.5) \\
\sigma_k &= \mu_k \cdot \rho_k \\
\end{align*}
\]
Else:
\[
\begin{align*}
\mu_k &= \mu_{k-1} \\
\sigma_k &= \sigma_{k-1} \\
\end{align*}
\]

Jobs generated according to this process are temporally correlated, and therefore not independent and identically distributed. Training data consists of 5000 trajectories of length 1000, each of which was randomly assigned a policy from Table 5.

Finally, Table 5 describes the 16 policies used in these experiments.

### C.2 Training setup

As before, we use MLP as the neural network structures for CausalSim models and the SLSim model and Table 4 is a comprehensive list of all hyperparameters used in training.
| Model          | Hyperparameter          | Value                  |
|---------------|-------------------------|------------------------|
|               | Hidden layers           | (128, 128)             |
|               | Hidden layer Activation function | Rectified Linear Unit (ReLU) |
|               | Output layer Activation function | Identity mapping |
|               | Optimizer               | Adam [29]              |
|               | Learning rate           | 0.001                  |
|               | $\beta_1$               | 0.9                    |
|               | $\beta_2$               | 0.999                  |
|               | $\epsilon$              | $10^{-8}$              |
|               | Batch size              | $2^{19}$               |
| **SLSim**     |                         |                        |
| **CausalSim** | $\kappa$                | {1, 5, 10, 15, 20, 25} |
|               | Training iterations (num_train_it) | 5000                  |
|               | num_disc_it             | 10                     |
| **SLSim**     | Training iterations     | 10000                  |

Table 2: Training setup and hyperparameters for the real-world ABR experiment
| Policies | Hyperparameter | Value | Used as source | Used as left out |
|----------|---------------|-------|----------------|-----------------|
| BBA      | Cushion       | 5     | ✓              | ✓               |
|          | Reservoir     | 10    |                |                 |
| BOLA-BASIC v1 | $V$ | 0.71 (Computed using puffer formula) | ✓               | ✓               |
|          | $\gamma$     | 0.22 (Computed using puffer formula) |             |                 |
|          | Utility function | ln(chunk sizes) (As used in BOLA paper [45]) |             |                 |
| Random   | -             | -     | ✓              | ✓               |
| BBA-Random mixture 1 | Cushion | 5 | ✓ | ✓ |
|          | Reservoir     | 10    |                |                 |
|          | Random choices | 50% |             |                 |
| BBA-Random mixture 2 | Cushion | 10 | ✓ | ✓ |
|          | Reservoir     | 20    |                |                 |
|          | Random choices | 50% |             |                 |
| MPC      | Lookback length | 5 | ✓ | ✓ |
|          | Lookahead length | 5 |             |                 |
|          | Rebuffer penalty | 4.3 |             |                 |
| Rate-based | Throughput estimate | Harmonic mean |             |                 |
| Optimistic Rate-based | Lookback length | 5 | ✓ | ✓ |
|          | Throughput estimate | Max |             |                 |
| Pessimistic Rate-based | Lookback length | 5 | ✓ | ✓ |
|          | Throughput estimate | Min |             |                 |

Table 3: ABR algorithms used in the synthetic ABR experiments.
### Table 4: Training setup and hyperparameters for the synthetic ABR experiment.

| Model          | Hyperparameter               | Value                        |
|----------------|-----------------------------|------------------------------|
| SLSim (1 network), CausalSim (3 networks) | Hidden layers                | (128, 128)                  |
|                | Hidden layer Activation function | ReLU                        |
|                | Output layer Activation function | Identity mapping           |
|                | Optimizer                    | Adam [29]                    |
|                | Learning rate                | 0.0001                       |
|                | $\beta_1$                   | 0.9                          |
|                | $\beta_2$                   | 0.999                        |
|                | $\varepsilon$               | $10^{-8}$                    |
|                | Batch size                   | $2^{14}$                     |
| CausalSim      | $\kappa$                    | $\{0.01, 0.1, 1, 10, 100\}$ |
|                | Training iterations (num_train_it) | 10000                      |
|                | num_disc_it                 | 10                           |
| SLSim          | Training iterations          | 10000                       |

### Table 5: Scheduling policies used in the load balancing experiment.

| Policies                             | Description                                                             | Used as source | Used as left out |
|--------------------------------------|-------------------------------------------------------------------------|----------------|------------------|
| Server limited policy (8 variations) | Randomly assign to only two servers                                    | ✓              | ×                |
| Shortest queue                       | Assign to server with smallest queue                                    | ✓              | ✓                |
| Power of $k$ ($k \in \{2,3,4,5\}$) | Poll queue lengths of $k$ server and assign to shortest queue          | ✓              | ✓                |
| Oracle optimal                       | Normalize queue sizes with server rates and assign to shortest normalized queue | ✓              | ✓                |
| Tracker optimal                      | Similar to oracle, but estimates server rates with historical observations of processing times | ✓              | ✓                |
| Model         | Hyperparameter                      | Value          |
|--------------|-------------------------------------|----------------|
| SLSim (1 network), CausalSim (3 networks) | Hidden layers              | (128, 128)     |
|              | Hidden layer Activation function    | ReLU           |
|              | Output layer Activation function    | Identity mapping |
|              | Optimizer                           | Adam [29]      |
|              | Learning rate                       | 0.0001         |
|              | $\beta_1$                           | 0.9            |
|              | $\beta_2$                           | 0.999          |
|              | $\epsilon$                          | $10^{-8}$      |
|              | Batch size                           | $2^{14}$       |
| CausalSim    | $\kappa$                            | {0.01, 0.1, 1, 10, 100} |
|              | Training iterations (num_train_it)  | 10000          |
|              | num_disc_it                         | 10             |
| SLSim        | Training iterations                 | 10000          |

Table 6: Training setup and hyperparameters for the load balancing experiment.