CausalSim: A Causal Framework for Unbiased Trace-Driven Simulation

Abdullah Alomar, Pouya Hamadanian, Arash Nasr-Esfahany, Anish Agarwal, Mohammad Alizadeh, and Devavrat Shah, *MIT*

https://www.usenix.org/conference/nsdi23/presentation/alomar

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CausalSim: A Causal Framework for Unbiased Trace-Driven Simulation

Abdullah Alomar* MIT aalomar@mit.edu
Pouya Hamadanian* MIT pouyah@mit.edu
Arash Nasr-Esfahany* MIT arashne@mit.edu
Anish Agarwal MIT anish90@mit.edu
Mohammad Alizadeh MIT alizadeh@mit.edu
Devavrat Shah MIT devavrat@mit.edu

Abstract

We present CausalSim, a causal framework for unbiased trace-driven simulation. Current trace-driven simulators assume that the interventions being simulated (e.g., a new algorithm) would not affect the validity of the traces. However, real-world traces are often biased by the choices algorithms make during trace collection, and hence replaying traces under an intervention may lead to incorrect results. CausalSim addresses this challenge by learning a causal model of the system dynamics and latent factors capturing the underlying system conditions during trace collection. It learns these models using an initial randomized control trial (RCT) under a fixed set of algorithms, and then applies them to remove biases from trace data when simulating new algorithms.

Key to CausalSim is mapping unbiased trace-driven simulation to a tensor completion problem with extremely sparse observations. By exploiting a basic distributional invariance property present in RCT data, CausalSim enables a novel tensor completion method despite the sparsity of observations. Our extensive evaluation of CausalSim on both real and synthetic datasets, including more than ten months of real data from the Puffer video streaming system shows it improves simulation accuracy, reducing errors by 53% and 61% on average compared to expert-designed and supervised learning baselines. Moreover, CausalSim provides markedly different insights about ABR algorithms compared to the biased baseline simulator, which we validate with a real deployment.

1 Introduction

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

Trace-driven simulation is a widely used method for evaluating new ideas in systems. In contrast to full-system simulation (e.g., NS3 [31]), which requires detailed knowledge of system characteristics (e.g., topology, traffic patterns, hardware details, etc.), trace-driven simulation does not model all components of a system. Instead, it focuses on simulating one (or a few) components of interest, where we wish to experiment with an *intervention*, e.g., a new design, algorithm, or architectural choice. To account for the effect of the remaining components that are not simulated, we collect a trace capturing their behavior and replay it while simulating the component of interest with the proposed intervention.

The key assumption here is that the interventions would not affect the trace being replayed, which we refer to as the *exogenous trace* assumption. If this assumption does not hold, replaying the trace is invalid and could lead to incorrect simulation results. This problem has been referred to as *bias* in trace-driven (or data-driven) simulation [15, 37].

It is difficult to guarantee the exogenous trace assumption in traces collected from real-world systems. Consider, for example, trace-driven simulation of adaptive bitrate (ABR) algorithms [35, 50, 63, 75]. It is common to use network throughput traces from real video streaming sessions on Internet paths [38, 75]. However, the throughput achieved when the player downloads a video chunk is caused by certain *latent* properties of the network path (e.g., the underlying bottleneck capacity, the number and type of competing flows, etc.), as well as the particular choices made by the ABR algorithm (the bitrate chosen for each chunk). In other words, the trace data reflects the combined effect of these two causes and is biased by the ABR algorithms used during trace collection. To simulate a new algorithm, we need to tease apart the effect of the two causes, and predict how the trace would have changed under the decisions of the new algorithm.

We present CausalSim, a causal framework for unbiased trace-driven simulation. CausalSim relaxes the *exogenous trace* assumption by explicitly modeling the fact that interventions can affect trace data. Using traces collected from a *randomized control trial* (RCT) under a fixed set of algorithms, it infers both the latent factors capturing the underlying conditions of the system and a causal model of its dynamics, including the unknown relationship between latents, algorithm decisions, and observed trace data. To simulate a new algorithm, CausalSim first estimates the latent factors at every time step of each trace. Then, it uses the estimated latent factors to predict the alternate evolution of the trace, actions, and observed variables of the component of interest, under the same latent conditions that were present when the trace was collected. This two-step process allows CausalSim to remove the bias in the trace data when simulating new algorithms.

*Equal contribution

CausalSim provides two benefits: (*i*) it improves the accuracy of trace-driven simulation when the intervention could affect (in possibly subtle ways) the trace data; (*ii*) it enables trace-driven simulation of systems where defining an exogenous trace is not possible and therefore standard trace-driven simulation is not applicable. We evaluate both settings in this paper, by simulating ABR and heterogeneous server load balancing algorithms as examples for cases (*i*) and (*ii*) respectively.

CausalSim requires training data from an RCT. Large network operators have increasingly invested in RCT infrastructure to evaluate new ideas, but due to their low throughput and risk of disruptions or SLA violations [42], they can afford to evaluate only a fraction of proposed ideas in RCTs. CausalSim greatly extends the utility of RCT data by learning a model that can simulate a wide range of algorithms using traces from a fixed set of algorithms. Periodically or whenever an operator believes the underlying system characteristics have changed significantly, they can collect fresh data using an RCT (again, with the same fixed set of algorithms) to retrain CausalSim.

CausalSim's design begins with the observation that unbiased trace-driven simulation can be viewed as a matrix (or tensor) completion problem [9, 14]. Consider a matrix M of traces (it is a tensor if traces are higher dimensional), with rows corresponding to possible actions and columns corresponding to different time steps in the trace data. For each column, the entry for one action is "revealed"; all other entries are missing. Our task can be viewed as recovering the missing entries.

A significant body of work has shown that it is possible to recover a matrix from sparse observations under certain assumptions about the matrix and the pattern of missing data. Roughly speaking, the typical assumptions that make recovery feasible are that the matrix has low rank, the entries revealed are chosen at random, and that enough entries are revealed. Low-rank structure is prevalent in many real-world problems [69] and has also been observed in network measurement data [16, 43, 44, 60]. But unfortunately the other two assumptions do not hold in our problem. As we detail in §4.3, one observed entry per column is below the information-theoretic bound for low-rank matrix completion (even for rank $r=1$). Moreover, not only are the entries revealed in our problem not random, they depend on other entries of the matrix, since the actions are being taken by algorithms based on observed variables.

To overcome these challenges, CausalSim exploits two key insights. First, it assumes a causal model (§3) where the latent factors are exogenous and are not affected by the interventions we want to simulate in the component of interest. This *exogenous latent* assumption relaxes (and is therefore implied by) the exogenous trace assumption in standard trace-driven simulation. For example, in ABR, it says that underlying factors like the bottleneck link speed on a network path are not affected by a user's ABR algorithm, whereas ABR decisions can impact the trace that user *observes* (i.e., the achieved throughput).

Second, CausalSim uses a basic property of trace data collected via an RCT. Since the assignment of an algorithm to a trace is completely random in an RCT, the distribution of latent factors should be the same for the traces obtained using different algorithms, i.e., the latent distribution is *invariant* to the algorithm. We provide conditions on the RCT data (e.g., in terms of the number and diversity of algorithms) that guarantee recoverability of the low-rank matrix using this invariance property (§4.2), and we operationalize this idea in a practical learning method that exploits the invariance using an adversarial neural network training technique (§5).

We evaluate CausalSim on two use cases, ABR and server load balancing, with both real-world and synthetic datasets, and further verify CausalSim's predictions with a test in the wild on the Puffer [71] video streaming testbed. Our main findings are:

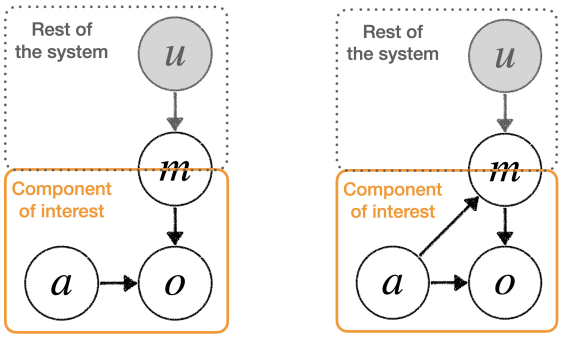
1. We use CausalSim to debug and improve an ABR algorithm, BOLA1 [53, 63]. In a ten month experiment on Puffer [71], BOLA1 exhibited high stalling compared to BBA [35], with slightly better quality. Using CausalSim, we tune BOLA1's parameters via Bayesian Optimization and deploy our improved version on Puffer. We show that it improves the stall rate of this well-known algorithm by 2.6×, achieving 0.7× the stall rate of BBA with similar perceptual quality. The expert-designed baseline simulator that ignores bias predicts the exact opposite: that the new variant should stall 1.34× the stall rate of BBA. This case study shows that removing bias is crucial to draw accurate conclusions from trace-driven simulation.
2. Evaluation of CausalSim on more than ten months of real data from Puffer shows that CausalSim's error in stall rate prediction is bounded to 28%, while expert-designed and standard supervised learning baselines have errors in the range of 49–68% and 29–187% respectively. Similar observations are also made for perceptual quality metrics and buffer occupancy levels.
3. CausalSim opens up new avenues to apply trace-driven simulation to systems where the exogenous trace assumption is invalid. Using a synthetic environment modeling a heterogeneous server load balancing problem, we show how CausalSim reduces average simulation error by 5.1×, a stark improvement compared to a baseline simulator with a median error of 124.3%.

This work does not raise any ethical issues. Our code is available at https://github.com/CausalSim/Unbiased-Trace-Driven-Simulation.

2 Motivation

2.1 Bias in Trace-Driven Simulation

Trace-driven simulation is a widely used technique to design and evaluate systems. Unlike full-system simulation, it focuses on simulating one (or a few) components of the system while capturing the effect of remaining components by replaying a trace. For example, to simulate new ABR algorithms, it is common to replay network throughput traces from real Internet



(a) Trace-driven simulation (b) CausalSim

Figure 1: CausalSim relaxes the *exogenous trace* assumption in standard trace-driven simulation.[1]

paths in a simulator modeling only the video player/server.

As we alluded to earlier, the key assumption here is that the interventions being simulated would not affect the trace being replayed; otherwise, replaying the trace would be invalid. We refer to this as the *exogenous trace* assumption, and it is central to standard trace-driven simulation. Figure 1a is a visual depiction of the exogenous trace assumption. In the figure, a represents the intervention we want to simulate; for example, the actions taken by a new algorithm. o is the observed state of the component being simulated. u represents the *latent* state of the rest of the system, which we do not observe or simulate. Finally, m is the trace, which captures the behavior of the other components.[2] The existence of each edge represents a causal effect. For example, the trace m and intervention a both affect o. Note the absence of the edge from a to m, which implies that the intervention cannot affect the trace (the exogenous trace assumption).

The simulator designer must define the trace carefully to meet this assumption. But what happens if it does not hold, i.e., there exists an edge from a to m (as in Figure 1b)? Ignoring the violation of *exogenous trace* assumption leads to biased simulation outcomes, as we will see next.

2.2 An Example Using Real-world Traces

In this section, we use more than ten months of real-world data from Puffer [71], a recently deployed system for experimenting with video streaming protocols, to illustrate the issue of bias in trace-driven simulation.

Puffer collects data from a continual Randomized Control Trial (RCT) that tests several Adaptive Bit Rate (ABR) algorithms. In the period of interest (July 27, 2020 – June 2, 2021), the tested algorithms include Buffer-Based Algorithm (BBA) [35], two versions of BOLA-BASIC (henceforth called BOLA) [63][3], and two versions of an algorithm called Fugu developed by the Puffer authors. The dataset includes more than 56 million chunk downloads from more than 230 thousand streaming sessions, totaling 3.5 years of streamed videos. For each streaming session, it provides logs of the chosen chunk sizes, available chunk sizes, achieved chunk download throughputs, and playback buffer levels.[4]

Consider a typical trace-driven simulation scenario, where we wish to simulate a new ABR algorithm using traces from previous video streaming sessions. We define such a task on the Puffer data as follows. We let one of the algorithms, say BBA, be the algorithm that we wish to simulate. We leave out the data for this algorithm and ask whether it is possible to predict its performance using the other algorithms' traces. In evaluating a new ABR algorithm, we may be interested in various performance measurements, e.g. buffer occupancy, rebuffering rate, chosen bitrates, etc. Here, we focus on predicting the behavior of playback buffer occupancy, which is one of the key indicators of an ABR algorithm's behavior [35].

The goal of trace-driven simulation is to predict the trajectory of the system (e.g., buffer, bitrates, etc.) for one algorithm in *the same underlying conditions* that were present when a trace was collected using a different algorithm. When simulating algorithm B based on a trace collected using algorithm A, we will refer to A as the "source" algorithm and to B as the "target" algorithm.

It is generally not possible to evaluate the accuracy of individual simulated trajectories using real-world data, because we do not have ground truth trajectories for the target algorithm under the same exact network conditions that were present when running the source algorithm. However, since the Puffer data was obtained using an RCT, we can evaluate predictions about *distributional* properties of the target algorithm, such as the distribution of the buffer occupancy achieved by the algorithm over the population of network paths present in the RCT.

To summarize, our task is: *predict the distribution of the buffer occupancy for the users assigned to BBA (the target algorithm) in the Puffer dataset, using only the data from the other (source) algorithms.*

2.2.1 Simulation via Expert Modeling (ExpertSim)

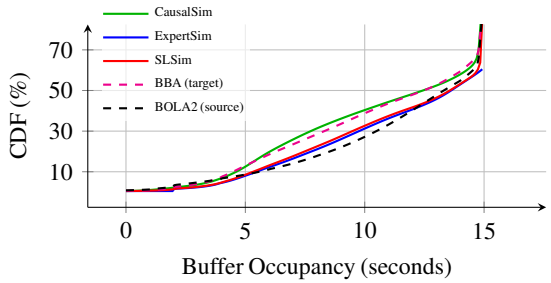
As our first strawman, we build a simple trace-driven simulator (ExpertSim) using our knowledge of how an ABR system works. ExpertSim models the playback buffer dynamics for each *step*, where a step corresponds to one ABR decision and

[1] In general, a and u can be correlated. For example, they can both depend on prior latent conditions of the system. In ABR, for instance, recent latent path conditions are correlated with current path conditions (u), and also affect the action taken by the ABR algorithm (a). Correlation of a and u, however, does not imply a causal relationship between them. In particular, our model assumes *exogenous latents*, i.e. a does not affect u.

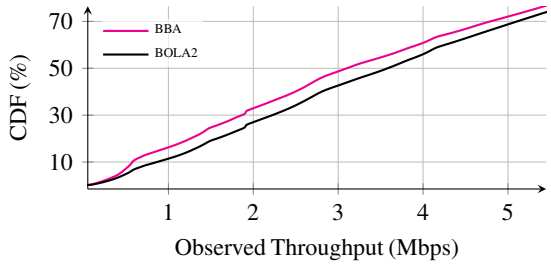
[2] Variables in Fig. 1a can be multidimensional and vary with time.

[3] BOLA1 and BOLA2 are variations on BOLA adjusted to target the SSIM quality metric instead of bitrate [53]. They pursue different objective functions and use different principles for hyperparameter adjustment.

[4] We use 'slow stream' logs (by Puffer's definition, streams with TCP delivery rates below 6Mbps) available on the Puffer website [1].



(a)



(b)

Figure 2: **(a)** CausalSim is accurate in predicting buffer level distribution of BBA users, while baseline simulators' predictions are similar to BOLA2 users. **(b)** Distribution of achieved throughput is different in BBA and BOLA2 users.

the download of a single video chunk. Let $\hat{c}_t$ be the throughput achieved in step t (for the t^{th} chunk) of a particular video streaming session using, say, the BOLA2 algorithm. To simulate BBA for the same user, ExpertSim assumes that the user would achieve the same throughput $\hat{c}_t$ in each step under the BBA algorithm as well. In other words, it assumes that ABR decisions do not affect the observed network throughput (the exogenous trace assumption). Under this assumption, ExpertSim models the evolution of the video playback buffer as follows. Let b_t be the buffer level at the beginning of step t (before the download of chunk t), r_t be the bitrate chosen in step t, and s_t be the size of the t^{th} chunk implied by the chosen bitrate. Then the buffer at the end of step t is derived as: $b_{t+1} = \max(0, b_t - s_t/\hat{c}_t) + T$, where T is the chunk duration.[5] Although simple, the assumption that throughput is an exogenous property of a network path is common in modelling ABR protocols. For example, both FastMPC [75] and FESTIVE [38] assume that the observed throughput does not depend on the chosen bitrate.

Figure 2a shows the true distribution of buffer level for BOLA2 and BBA users in the Puffer dataset (the two dashed lines), as well as the distribution *predicted* by running BBA on the traces collected from BOLA2 users using ExpertSim (solid blue line). The predictions are inaccurate: the buffer distribution generated by ExpertSim is more similar to the buffer distribution of BOLA2 users (the source algorithm) than the buffer distribution of BBA users (the target algorithm).

[5] The complete buffer dynamic equation is slightly more complex to handle cases with full buffers. Refer to §C.1 in the appendix for further clarification.

2.2.2 Simulation via Supervised Learning (SLSim)

Perhaps the simple model of buffer dynamics in ExpertSim does not accurately reflect the actual system behavior. As a next attempt, we turn to machine learning and try to learn the system dynamics from data. Specifically, we use supervised learning to train a Neural Network (NN) that models the step-wise dynamics of the system. This fully connected NN includes 2 hidden layers, each with 128 ReLU activated neurons. For each timestep t, the NN takes as input the buffer level before downloading the t^{th} chunk b_t, the achieved throughput $\hat{c}_t$ for chunk t, and the chunk size s_t (which depends on the birate chosen by ABR). The NN outputs the download time of the t^{th} chunk, and the resulting buffer level b_{t+1}. We train the NN to minimize the prediction error on our dataset. To avoid information leaking, we exclude the logs for BBA from the training data.

Figure 2a shows the predicted buffer level distribution via this approach (SLSim) for BBA. As with ExpertSim, we use the traces collected from BOLA2 users as the source algorithm. The results are similar to ExpertSim; once again, the predicted buffer distribution is closer to that of BOLA2 than BBA.

2.2.3 What Went Wrong?

To understand the limitations of ExpertSim and SLSim, we plot the distribution of achieved per-chunk throughput for users assigned to BOLA2 and BBA in Figure 2b. Since algorithm selection is completely random, we would expect inherent network path properties such as bottleneck link capacity to have the same distribution for users assigned to different ABR algorithms. However, such an invariance should not be expected for achieved throughput, because even on the same path different ABR algorithms could achieve different throughput. For example, since congestion control protocols take time to discover available bandwidth (e.g., in slow start) or converge to their fair share rate when competing against other flows, an ABR algorithm that tends to choose lower bitrates (and hence download less data per chunk) may achieve less throughput than an ABR algorithm that picks higher bitrates [34, 64]. We can see this behavior in the Puffer dataset. The achieved throughput for BOLA2 and BBA is clearly different in Figure 2b.

This confirms that ABR algorithms cause a bias in the measured throughput traces, and the exogenous trace property does not hold. To perform accurate trace-driven simulation, we need to account for this bias when simulating new ABR algorithms.

2.3 Causal Inference to the Rescue!

If the traces were the *underlying network capacity* when each chunk was downloaded (rather than the achieved throughput), the exogenous trace assumption would hold and our problem would be simple. First, we would learn the relationship between network capacity and achieved throughput for different ABR actions using our data. Then, to simulate BBA for a given trace, we would start with the network capacity

at each step of the trace and predict the achieved throughput taking into account the bitrate chosen by BBA in that step. This would then allow us to predict how the buffer evolves. This works because unlike achieved throughput, underlying capacity is an exogenous property of a network path and is not affected by the ABR actions.

However, underlying network capacity is a *latent* quantity — we do not observe it in our traces. The key challenge is therefore to *infer* such latent quantities from observational data. Concretely, in our running example, we wish to estimate the latent factors like network capacity in each step of a trace, using observations such as the bitrate, the chunk size, the achieved throughput, etc.[6]

Inferring such *latent confounders* and using them for counterfactual prediction is the core issue in the field of causal inference [57, 58]. In this paper, we develop CausalSim, a causal framework for unbiased trace-driven simulation. CausalSim relaxes the exogenous trace assumption in trace-driven simulation. It explicitly models the fact that interventions can affect trace data (the edge from a to m in Figure 1b), and infers both the latent factors and a causal model of the system dynamics. This allows CausalSim to correct for the bias in trace data when simulating an intervention. As an illustration, Figure 2a shows the predicted buffer occupancy distribution when simulating BBA on the traces of users assigned to BOLA2, using CausalSim. CausalSim matches the ground-truth distribution for BBA much more accurately than the alternatives.

3 Model and Problem Statement

3.1 Causal Model

Consider the following discrete-time dynamical model[7] corresponding to Figure 1b:

$$m_t = \mathcal{F}_{\text{trace}}(a_t, u_t), \tag{1}$$

$$o_{t+1} = \mathcal{F}_{\text{system}}(o_t, m_t, a_t). \tag{2}$$

Here, t denotes the time index, m_t is the trace, a_t is the intervention, u_t is the latent factor, and o_t is the observed state of the component of interest. The function $\mathcal{F}_{\text{trace}}$ models the effect of interventions on the trace (which traditional methods ignore), and $\mathcal{F}_{\text{system}}$ models the dynamics of the component of interest. When the intervention changes an algorithm in the component of interest, a_t can be viewed as the action taken by that algorithm at time t.

We assume that interventions do not affect the internal state of the rest of the system, i.e., that the latent factors are exogenous. This assumption is implicit in the dynamical system equations, and also visualized in Figure 1b by the absence of the edge from a to u. Note that this is a strict relaxation of the exogenous trace assumption in standard trace-driven simulation. There, the trace itself is assumed to be unaffected by intervention, which also implies exogenous latent factors.

In our running ABR example, we want to simulate the video player and server (components of interest) without precisely modeling the entire network path (the rest of the system). Each time step t corresponds to the download of a new chunk, and u_t represents latent network conditions during that transmission, e.g., bottleneck link speed, number of flows sharing the same network path, type of congestion control used by competing flows, etc. At each time step, the ABR algorithm chooses a bitrate a_t, which together with u_t generate m_t, the *achieved* throughput when downloading a chunk. Typically, latent network conditions are exogenous factors, beyond the impact of a particular user's actions. For instance, the bottleneck link speed and type of congestion control that competing flows use, are not affected by the actions of the ABR algorithm.

Note that the achieved throughput depends on the ABR action as well as the latent network conditions. Equation (1) captures this relationship and is the source of the bias induced by the ABR algorithm, which we demonstrated in §2.2.3.

When is the model applicable? The causal model applies in any trace-driven simulation setting where the trace may be impacted by interventions. Examples include:

- Job scheduling, where we wish to simulate a workload's performance under different types of machines. The trace is the job performance (e.g., runtime), interventions are the scheduling decisions, and latent factors are intrinsic properties of each job (e.g., compute intensity) or latent aspects of the machines such as collocated interfering workloads.

- Network simulation, where we wish to simulate how some aspect of network's design (e.g., congestion control, packet scheduling, traffic engineering, etc.) impacts application performance. The trace is an application's traffic pattern, the intervention is the network design, and latent factors are the internals of the application that dictate its traffic demand.

In some cases, like our running ABR example, the exogenous trace assumption may not hold exactly but still be roughly valid.[8] Here, CausalSim removes bias and improves simulation accuracy. But in certain problems, ignoring the effect of interventions is meaningless. For example, consider scheduling or load balancing on heterogeneous machines (e.g., with different hardware capabilities). Given a trace of job performance on specific machines, it isn't possible to merely replay the trace for new machine assignments. In

[6] For simplicity, we only mention network capacity here, but other latent path conditions like the number of competing flows could also affect achieved throughput and the same reasoning applies to them.

[7] This model is similar to a special type of Partially Observable Markovian Decision Processes (POMDPs) in which the unobserved part of the state is exogenous [51].

[8] Even in these cases, these subtly biased simulations can produce entirely incorrect conclusions (§6.2).

such problems, CausalSim enables trace-driven simulation by explicitly modeling the effect of interventions on the trace.

When is the model invalid? Our causal model relaxes the exogenous trace assumption but still requires *exogenous latents*, i.e. that the latents are unaffected by the intervention. This won't hold in all systems. For example, we cannot model the effect of network routing policies (e.g., BGP) on observed video streaming throughput in this way, since changing the path would change the latent network conditions that impact a video stream. Another example is simulating the effect of a CPU feature like the branch predictor on instruction throughput. Here, we can't model the state of the instruction/-data caches as an exogenous latent factor, since changing the branch predictor can change their internal state significantly.

Overall, a simulation designer needs to reason about the causal structure of observed and latent quantities to define the appropriate model in the form of Equations (1) and (2). However, the designer does not need to precisely specify the meaning of the latents or the dynamics (the functions $\mathcal{F}_{\text{trace}}$ and $\mathcal{F}_{\text{system}}$). CausalSim learns both from observational data.

3.2 Problem Formulation

We are given N trajectories, collected using K specific policies.[9] Let H_i be the length of trajectory $i \in \{1,\dots,N\}$. For trajectory i, we observe $(m_t^i, o_t^i, a_t^i)_{t=1}^{H_i}$. We assume that trajectories are generated using an RCT, i.e., that each trajectory is assigned to one of the K policies at random.

Our goal is to estimate the observations under an arbitrary given intervention (e.g., a new algorithm) for each of the N trajectories. Let $\{u_t^i\}_{t=1}^{H_i}$ be the exogenous latent factors for trajectory i. Formally, for any given trajectory i and given a sequence of actions $\{\tilde{a}_t^i\}_{t=1}^{H_i}$, starting with observation o_1^i and under the same sequence of latent factors $\{u_t^i\}_{t=1}^{H_i}$, we wish to estimate the counterfactual observations $\{\tilde{o}_t^i\}_{t=1}^{H_i}$ that are consistent with Equations (1) and (2).

This is a counterfactual estimation problem since it requires (i) estimating *latent* $\{u_t^i\}_{t=1}^{H_i}$ factors for observed trajectory i and using them along with the counterfactual actions $\{\tilde{a}_t^i\}_{t=1}^{H_i}$ to predict the counterfactual trace $\{\tilde{m}_t^i\}_{t=1}^{H_i}$ consistent with Equation (1), and then (ii) using the counterfactual trace and actions to predict counterfactual observations $\{\tilde{o}_t^i\}_{t=1}^{H_i}$ consistent with Equation (2).

For (ii), learning $\mathcal{F}_{\text{system}}$ is a supervised learning task because its inputs, (o_t^i, m_t^i, a_t^i), and output, o_{t+1}^i, are fully observed. If $\{u_t^i\}_{t=1}^{H_i}$ was observed, then (i) would also boil down to learning $\mathcal{F}_{\text{trace}}$ in a supervised manner. *It is the lack of observability of* $\{u_t^i\}_{t=1}^{H_i}$ *that makes our simulation task extremely challenging. In short, we are left with (i), the task of estimating* $\{\tilde{m}_t^i\}_{t=1}^{H_i}$ *and learning* $\mathcal{F}_{trace}$.

[9] We use policy and algorithm interchangeably in this paper.

4 CausalSim: Theoretical Insights

This section describes the theory behind CausalSim. We discuss how to operationalize this theory in a practical learning algorithm in §5. We begin by casting counterfactual estimation as a challenging variant of the matrix completion problem [14]. We then formalize conditions that allow us to complete the matrix using a certain distributional invariance property that is present in data collected in an RCT.

4.1 Counterfactual Estimation as Matrix Completion

Recall from §3.2 the task of estimating the counterfactual trace $\{\tilde{m}_t^i\}_{t=1}^{H_i}$ consistent with Equation (1). In this section, we pose this task as a variant of the classical matrix completion problem. For simplicity, let action a_t^i be one of the finitely many options $\{1,\dots,A\}$ for some $A \geq 2$. Imagine an A by U matrix M, where rows correspond to A potential actions, and columns corresponds to $U = \sum_{i=1}^{N} H_i$ latent factors (u_t^i for different choices of i and t) in the dataset. To order the columns, we may index u_t^i as a tuple $(i,\ t)$ and order these tuples in lexicographic order. The matrix M is called the potential outcome matrix in the causal inference literature [61].

At the t^{th} step of the i^{th} trajectory, we observe $m_t^i = \mathcal{F}_{\text{trace}}(a_t^i, u_t^i)$, which is the entry in M in the row corresponding to a_t^i and the column corresponding to u_t^i. The counterfactual quantities of interest, $\tilde{m}_t^i = \mathcal{F}_{\text{trace}}(\tilde{a}_t^i, u_t^i)$ for $\tilde{a}_t^i \neq a_t^i$, are the missing entries in M in the same column. In summary, we observe one entry per column of the matrix M and we wish to estimate the missing values in the matrix.

The task of filling missing values in a matrix based on its partially observed entries is known as *Matrix Completion* [19], a topic that has seen tremendous progress in the past two decades [18, 20, 47]. However, standard matrix completion methods do not apply to our problem (see §4.3 for details).

We use a distributional invariance property of data collected using an RCT to complete the potential outcome matrix M. The key observation is that, in an RCT, the latent factors for trajectories collected under each of the policies will have the same distribution. For example, in Puffer's RCT, incoming users are assigned to an ABR algorithm at random. Therefore each ABR algorithm will "experience" the same distribution of underlying latent network conditions, which is precisely why we can compare their performance in the RCT. The same property helps us recover the matrix M, as we show next.

4.2 Exploiting RCT for Matrix Completion

We use a minimal non-trivial example to give intuition about how we can exploit an RCT for matrix completion, before stating our main theoretical result.

Consider a simple example where $A = 2$ and $U = 2n$, and the rank of potential outcome matrix M is equal to 1. Rank 1

implies that $M = au^T$ for some $a \in \mathbb{R}^2$ and $u \in \mathbb{R}^{2n}$ with $M_{\alpha,\beta} = a_\alpha \cdot u_\beta$.[10] Suppose we have $K = 2$ policies, where each policy always chooses only one of the two actions. Furthermore, we consider an RCT setting. That is, the distribution of latent factors across trajectories assigned to both policies should be the same.

Without loss of generality, we can re-order the columns of M so that the first n columns correspond to the latent factors of the trajectories assigned to policy 1, and the second n columns are those assigned to policy 2. Then the observed entries of matrix M appear as

$$\begin{bmatrix} M_{1,1} & M_{1,2} & \dots & M_{1,n} & \star & \dots & \star & \star \\ \star & \star & \dots & \star & M_{2,n+1} & \dots & M_{2,2n-1} & M_{1,2n} \end{bmatrix}$$

where $\star$ represents the missing values.

Let us consider recovering the missing observation $M_{2,1}$. For column 1, we know the observation under the first action, i.e. $M_{1,1}$. Due to rank 1 structure, we have

$$\frac{M_{2,1}}{M_{1,1}} = \frac{a_2 u_1}{a_1 u_1} = \frac{a_2}{a_1}. \tag{3}$$

Therefore, to find $M_{2,1}$ (and by a similar argument, to find all missing entries of M), we need to estimate the ratio $\frac{a_2}{a_1}$.

Due to the distributional invariance induced by RCT, the samples $u_1, \dots, u_n$ (which correspond to the latent factors encountered by policy 1) come from the same distribution as the samples $u_{n+1}, \dots, u_{2n}$ (which correspond to the latent factors encountered by policy 2), for large enough n. Thus, their expected value should be equal:

$$\frac{1}{n}\sum_{\beta=1}^{n} u_\beta \approx \frac{1}{n}\sum_{\beta=n+1}^{2n} u_\beta \tag{4}$$

Equation (4) implies

$$\frac{\sum_{\beta=1}^{n} M_{1,\beta}}{\sum_{\beta=n+1}^{2n} M_{2,\beta}} = \frac{\sum_{\beta=1}^{n} a_1 \cdot u_\beta}{\sum_{\beta=n+1}^{2n} a_2 \cdot u_\beta} \approx \frac{a_1}{a_2}. \tag{5}$$

This provides precisely the quantity of interest in Equation (3) based on the observed entries, enabling us to complete the matrix.

Formal Result. This simple illustrative example relied on a convenient observational pattern (based on policies that always choose one action) and rank 1 structure. But the idea can be generalized. If the trace includes D measurements, $M_{\alpha,\beta,\gamma} \in \mathbb{R}^{A \times U \times D}$ becomes a tensor rather than a matrix, where α, β, and γ index the actions, latent factors, and measurements, respectively. The following theorem provides conditions where completion is possible for a rank r tensor. For more details and the proof, refer to Appendix A.

Theorem 4.1. *We can recover all entries of M by only observing one $D-$dimensional element in each column (corresponding to one latent and action) if the following is satisfied:*

1. ***(Low-Rank Factorization)*** *M is a low-rank tensor (rank $= r$), i.e., it admits the following factorization: $M_{\alpha,\beta,\gamma} = \sum_{\ell=1}^{r} a_{\alpha\ell} u_{\beta\ell} z_{\gamma\ell}$.*
2. ***(Invertibility)*** *The factorization implies existence of a linear mapping from latent encoding to trace for each action. This linear mapping is invertible.*
3. ***(Sufficient measurements)*** *$D \geq r$.*
4. ***(Sufficient, Diverse Policies)*** *The number of policies $K \geq Ar$, and the matrix $\mathbf{S} \in \mathbb{R}^{Ar \times K}$ is full-rank where $\mathbf{S}_{w.D:(w+1).D,x} = \mathbb{E}[m|action\_index = w, policy\_index = x]\mathbb{P}(action\_index = w|policy\_index = x)$. Linear independence of columns of $\mathbf{S}$ can be interpreted as diversity among policies (Appendix A).*

4.3 Discussion

Why not standard tensor completion? Tensor completion methods [26, 41, 48, 78] make several assumptions. First, the tensor M must be (approximately) low rank, which CausalSim also requires. Low-rank structure holds in many real-world problems [69] and has been observed in network measurements, e.g., in traffic matrices [16, 43, 44, 60] and network distance (i.e., RTT) [46, 52, 66]. As an example of how it emerges in the problems we study in this paper, we use a simple model of congestion control in Appendix C.4 to provide intuition about low-rank structure in ABR data.

Second, the pattern of missing entries should be *random*. If the missing patterns is not random and depends on latent factors or the entries themselves [8], standard approaches have difficulty recovering the tensor. This assumption does not hold in trace-driven simulation. Revealed entries are determined by the actions taken by the policies, which often use recent observations to make their decisions (e.g., an ABR policy may use recent throughput measurements). Hence the revealed/missing entries in a column are not random and depend on the entries in previous columns.

Third, a sufficient number of entries need to be revealed. For example, when $D = 1$ (i.e., when M is a matrix), the information theoretic lower bound to on the number of revealed entries needed to recover M is $4Ur - r^2$ [39, 70]. Thus even for rank $r = 1$, it requires 4 entries per column, whereas only one entry per column is revealed in trace-driven simulation.

Since the second and third assumptions do not necessarily hold in our setup, we cannot use *existing* tensor completion methods. However, as we argued in §4.2, exploiting the additional problem structure imposed by RCT data can make tensor completion feasible in certain conditions.

Limitations of Theorem 4.1. The proof of Theorem 4.1 (Appendix A) provides an analytical method for recovering the tensor M that generalizes the procedure described for the simple example in §4.2. While this provides a theoretical basis for why tensor recovery is possible, the analytical approach is not practical. First, it relies on M being exactly rank r; if it is approximately rank r, we have found the calculation to be

[10] Note that for readability, we are abusing notation by overloading a and u to refer to both the action and latent, and their encodings in the factorization.

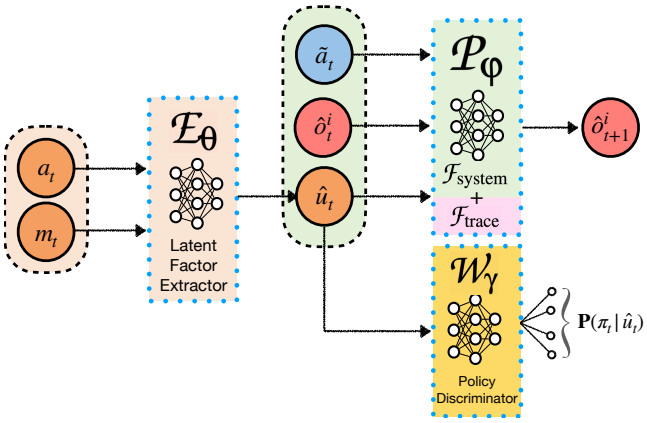


Figure 3: CausalSim Architecture

brittle. Second, it applies only to discrete action spaces. Third, it gives sufficient conditions for recovery, but they're not all necessary. One reason is that the analytical method uses only *mean* invariance, i.e. the fact that the mean of the latent factors is the same across all policies (as in Eq. (4)), even though RCT data has the stronger property that the entire *distribution* of latents does not depend on the policy. In the next section, we describe our practical implementation of CausalSim that uses learning techniques and NNs to overcome these limitations (at the expense of theoretical guarantees).

5 CausalSim: Algorithm

CausalSim builds upon the insights presented earlier but replaces the factorized model with a learning algorithm based on NNs. For ease of notation, we will drop the trajectory index for all variables in the dataset, e.g. we will refer to the latent factor $u^i_t : t \le H_i, i \le N$ as $u_t : t \le H$.

CausalSim architecture. As discussed, CausalSim aims to extract u_t and learn $\mathcal{F}_{\text{trace}}$ and $\mathcal{F}_{\text{system}}$ from observed trajectories $(o_{t+1}, o_t, m_t, a_t) : t < H$. Figure 3 summarizes CausalSim's algorithmic structure.

To extract latent factors, we use a NN that takes in a_t and m_t, and computes $\hat{u}_t$ (an estimate of u_t). To apply invariance on the extracted latents, i.e. distribution of u_t being the same regardless of the policy applied to it, we use a NN called the *Policy Discriminator*. This NN aims to predict the policy pertaining to that sample given $\hat{u}_t$, and if invariance is upheld, it will fail to do so. Unlike the analytical approach, the policy discriminator can enforce policy invariance on the entire latent distribution, potentially improving the accuracy of the estimate.

To calculate the counterfactual traces and observations, we need to learn $\mathcal{F}_{\text{trace}}$ and $\mathcal{F}_{\text{system}}$. However, we can simplify the learning problem by merging these two into one single combined function. Thus, we use a NN that takes in counterfactual actions $\tilde{a}_t$, observation o_t and estimated latent $\hat{u}_t$, and computes counterfactual observation $\tilde{o}_{t+1}$. Of course, we can explicitly use separate NNs for $\mathcal{F}_{\text{trace}}$ and $\mathcal{F}_{\text{system}}$ if we require access to the simulated trace ($\tilde{m}_t$) values.

Algorithm 1 CausalSim Training

1: initialize parameter vectors γ, θ, φ
2: initialize hyper-parameters *num_disc_it*, κ
3: initialize dataset $D \leftarrow \{(o_i, m_i, a_i, \pi_i)\}_{i=1}^m$ from an RCT
4: **for** each iteration **do**
5: ↑ Discriminator **for** *num_disc_it* **do**
6: sample minibatch $B \leftarrow \{(o_l, m_l, a_l, \pi_l)\}_{l=1}^b$
7: $u_l \leftarrow \mathcal{E}_\theta(m_l, a_l)$ for $l \in \{1, ...b\}$
8: $\mathcal{L}_{\text{disc}} \leftarrow \frac{1}{b}\Sigma_{l=1}^b \left[-\log \mathcal{W}_\gamma(\pi_l \mid u_l)\right]$
9: $\gamma = \gamma - \lambda_\gamma \cdot \nabla_\gamma \mathcal{L}_{\text{disc}}$
10: ↓ **end for**
11: ↑ Simulation Modules sample minibatch $B \leftarrow \{(o_{l+1}, o_l, m_l, a_l, \pi_l)\}_{l=1}^b$
12: $u_l \leftarrow \mathcal{E}_\theta(m_l, a_l)$ for $l \in \{1, ...b\}$
13: $\mathcal{L}_{\text{disc}} \leftarrow \frac{1}{b}\Sigma_{l=1}^b \left[-\log \mathcal{W}_\gamma(\pi_l \mid u_l)\right]$
14: $\mathcal{L}_{\text{pred}} \leftarrow \frac{1}{b}\Sigma_{l=1}^b \left[\left(o_{l+1} - \mathcal{P}_\varphi(o_l, a_l, u_l)\right)^2\right]$
15: $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{pred}} - \kappa \cdot \mathcal{L}_{\text{disc}}$
16: $\theta = \theta - \lambda_\theta \cdot \nabla_\theta \mathcal{L}_{\text{total}}$
17: ↓ Simulation Modules $\varphi = \varphi - \lambda_\varphi \cdot \nabla_\varphi \mathcal{L}_{\text{pred}}$
18: **end for**

Overall, CausalSim uses three NNs for counterfactual simulation; $\mathcal{E}_\theta$ as the *latent factor extractor*, W_γ as the *policy discriminator* and $\mathcal{P}_\varphi$ as the combination of $\mathcal{F}_{\text{trace}}$ and $\mathcal{F}_{\text{system}}$. Figure 3 depicts the structure. Training these NNs is quick; on an A100 Nvidia GPU, CausalSim's time to convergence on $56M$ data points ($230K$ streams) was less than 10 minutes, and each simulation step in inference (on CPU) takes less than $150\mu s$. A full inference run on the same volume of data takes less than 6 hours on a single CPU core and less than 20 minutes on 32 cores.

Training procedure. CausalSim's training procedure alternates between: (i) training the policy discriminator using a discrimination loss $\mathcal{L}_{\text{disc}}$; and (ii) training other modules using an aggregated loss $\mathcal{L}_{\text{total}}$. Algorithm 1 provides a detailed pseudo code of this training procedure.

Training the policy discriminator (Lines 5–10 in Algorithm 1). Distributional invariance means restricting the distribution of latent factors u to be identical across policies. To that end, we first use $\mathcal{E}_\theta$ to extract latents $\hat{u}_t$, and then search for invariance violations via a *discriminator* NN, a standard approach in the paradigm of adversarial learning [29, 68]. Specifically, the policy discriminator aims to predict the policy π_i that took action a_t from the estimated latent factor $\hat{u}_t$ (see Figure 3). Towards that, we use a cross-entropy loss to train the policy discriminator:

$$\mathcal{L}_{\text{disc}} = \mathbb{E}_B[-\log \mathcal{W}_\gamma(\pi \mid \hat{u})], \tag{6}$$

where the expectation is over the a sampled minibatch B from dataset D. We train the policy discriminator to minimize this loss, by repeating gradient decent *num_disc_it* times, as the

policy discriminator needs multiple iterations to catch up to changes in the latent factors.

Training simulation modules (Lines 11–17 in Algorithm 1). In this step, we need to impose consistency with observations, all while preserving the distributional invariance. Thus, we compute latent factors $\hat{u}_t$ with $\mathcal{E}_\theta$ and simulate the next step of the trajectory $\hat{o}_{t+1}$ with $\mathcal{P}_\varphi$. We use an aggregated loss to enforce consistency and invariance. This loss combines the negated discriminator loss with a quadratic consistency loss using a mixing hyper-parameter κ.

$$\mathcal{L}_{\text{total}} = \mathbb{E}_B\left[(o_{t+1} - \hat{o}_{t+1})^2\right] - \kappa\mathcal{L}_{\text{disc}}, \tag{7}$$

where the expectation is over the a sampled minibatch B from dataset D. Here, we used a quadratic loss function, but one could use any consistency loss fit to the specific type of variable (e.g. Huber loss, Cross entropy, ...).

Note the negative sign of discriminator loss, which means we train these NNs to maximize discriminator loss i.e., fool the discriminator to ensure policy invariance. If the extracted latent factors are policy invariant, the policy discriminator should do no better at its task than guessing at random.

Counterfactual estimation. To produce counterfactual estimates, as described above, the estimated latents $\hat{u}_t$ are extracted from observed data. Using the extracted latents factors, along with the learned combined function $\mathcal{P}_\gamma$, we start with o_1 and predict counterfactual observations $\hat{o}_{t+1}$, one step at a time.

6 Evaluation

We evaluate CausalSim's ability to do accurate counterfactual simulation (§6.1 and §6.3) using trace data from one real-world and one synthetic dataset. As a rigorous proof of concept, we debug and improve an ill-performing ABR policy with CausalSim (§6.2), and verify it through deployment on a public ABR testing infrastructure. Our baselines are as follows:

1. *ExpertSim*: Uses the analytical model described in §2.2.1.
2. *SLSim*: Uses a standard supervised-learning technique to learn system dynamics from data, as described in §2.2.2.

Finally, we show how CausalSim enables trace-driven simulation in problems where defining an *exogenous trace* is not straightforward and traditional trace-driven simulation is not applicable (§6.4). Further supporting experiments in the appendix provide more details about how CausalSim operates (§B.1, §B.2, §B.3, §B.4, §B.5, §B.7, §C.2, §C.3, §C.4 and §D.1).

6.1 Simulation Accuracy

We use CausalSim to predict the end performance of ABR policies, and compare them with ground truth data. We explore the same two metrics reported by Puffer to evaluate algorithms; 1) stall rate, which is the fraction of time a user spent rebuffering, i.e. paused and waiting for a new chunk to download; 2) average Structural Similarity Index Measure (SSIM) in decibels, which is a perceptual quality metric. Our ground truth data comes from public logs of 'slow streams' on Puffer. 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. Our dataset contains more than $230K$ trajectories from an RCT during July 2020 to June 2021, where five ABR algorithms (BBA, BOLA1, BOLA2, Fugu-CL, Fugu-2019) were evaluated. Exhaustive details of the setup and data can be found in §B.8.

6.1.1 Can CausalSim simulate a policy it has not seen?

We choose one of BBA, BOLA1, and BOLA2[11] as the new policy that we want to simulate, and call it the *target* policy. The remaining four policies are called *source* policies. Traces assigned to the four source policies comprise our training dataset, which we use for training CausalSim and the two baselines. The goal is to simulate the outcome of applying the target policy on trajectories assigned to any of the source policies.

Figure 4a plots the stall rate and SSIM in the simulated trajectories and ground truth, denoting each target policy with a different color. Four source policies give us four separate predictions per target policy and simulator. Each point depicts the average of these four predictions, and the intervals show the minimum and maximum among the four. For either metric, CausalSim is the most faithful to ground truth among all simulators. For instance, in stall rate, CausalSim's relative error spans $2-28\%$, while ExpertSim spans $49-68\%$ and SLSim spans $29-187\%$. CausalSim may not always predict the correct relative ordering among policies with close performance. For example, BOLA1 and BOLA2 (shown in orange and red) have similar performance in both stall rate and SSIM. CausalSim predicts that these policies are similar but it infers their relative ordering incorrectly. However, CausalSim avoids the large errors made by the baseline simulators. In absolute terms, its predictions are close to the ground truth.

CausalSim also has the most consistent predictions across different **source** policies, because it removes the biases of the source policies. As an example, we investigate all four simulation results for BOLA1 in Figure 4b. SLSim and ExpertSim's simulation results are only good when the source algorithm is BOLA2 (a similar algorithm to BOLA1 performance-wise). However, their predictions are far off from the ground truth for the other three source algorithms. CausalSim's simulation results, on the other hand, are all close to the ground truth target. Appendix §B.7 demonstrates the same observation for other target algorithms, i.e. BBA and BOLA2.

[11] We exclude Fugu as a test policy since we could not reproduce its logged actions (see §B.8).

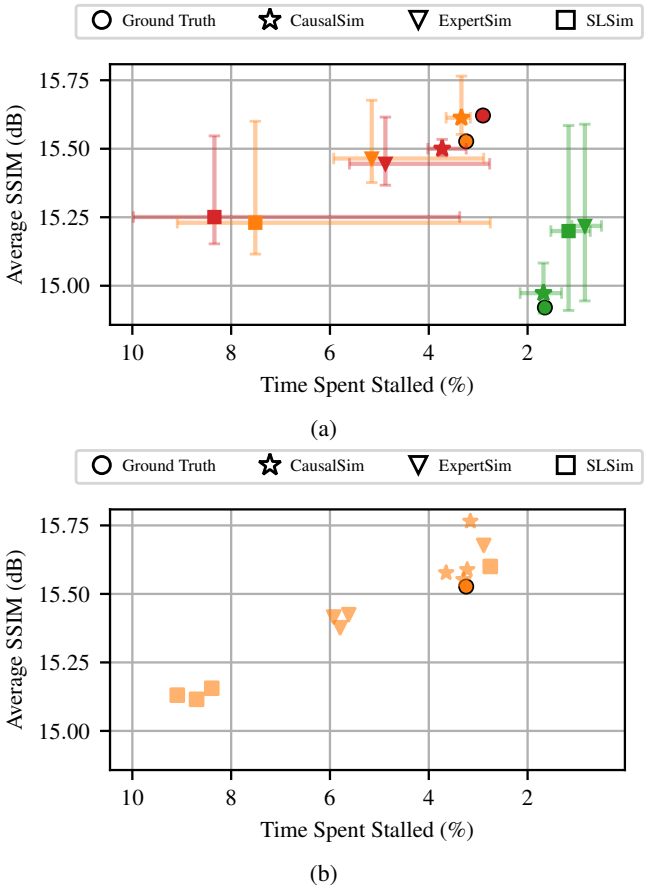


Figure 4: **(a)** In a real-world dataset of live video streaming, CausalSim is the most faithful, compared to traditional trace-driven (ExpertSim) or data-driven (SLSim) simulators. Colors indicate different **target** ABR algorithms. **(b)** Predictions for BOLA1, separated by the source policy. Each point indicates a different **source** ABR algorithm. ExpertSim and SLSim predictions carry over biases of the source data, while CausalSim mitigates the bias.

6.2 Case Study: CausalSim in the Wild

An accurate simulator allows researchers to debug and improve protocols without repeated and invasive deployments. We shall demonstrate this with CausalSim, by improving a well-known ABR policy, and verifying our findings with a real-world deployment on Puffer.

Recall that in the particular RCT we used in §6.1, five ABR algorithms (BBA, BOLA1, BOLA2, Fugu-CL, Fugu-2019) were evaluated. Figure 5 shows the result of this evaluation for BBA, BOLA1 and BOLA2, across 'slow streams'.[12] Similar to Figure 4a, the X-axis shows the stall rate, and the Y-axis is the average SSIM. BOLA1 exhibited 82% more rebuffering compared to BBA. A revised version of BOLA1, called BOLA2, was deployed alongside it, since the Puffer

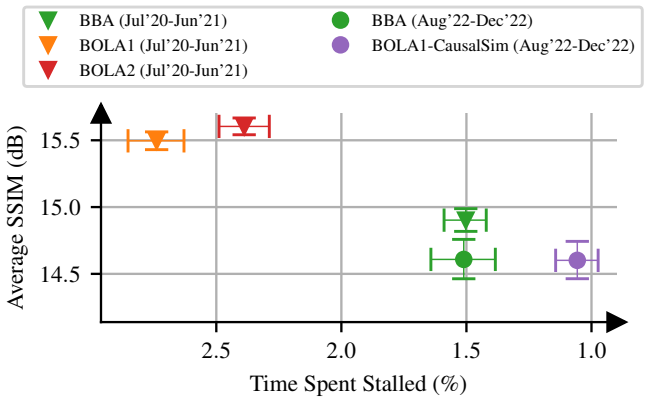


Figure 5: In an experiment preceding this work, BOLA1 exhibits high stalling. By deploying a BOLA1 variant in a later experiment **CausalSim improved the stall rate by** 2.6×, with comparable quality to BBA. User population is 'slow streams' and error bars denote 2.5%–97.5% confidence intervals.

team and the authors of BOLA believed the SSIM metric (in decibels) is incompatible with the protocol [53]. This new version had 12.8% less rebuffering and slightly higher quality, but still far too much stalling compared to BBA.

BOLA1 is an ABR policy with two hyperparameters, similar to BBA, and our hypothesis was that BOLA1 uses sub-optimal hyperparameters. To investigate this, we used the logged data pertaining to that plot along with CausalSim to exhaustively analyze the performance of BOLA1 and BBA for a range of hyperparameters. Using Bayesian Optimization[13], we explored the parameter space and created a Pareto frontier curve for each policy. During this process, we evaluated over 150 different algorithms in two days, which is achievable only in a simulator. Each curve demonstrates the trade-off between quality and stall rate in that policy. Figure 6 presents the curves, where the left and right plots show CausalSim and ExpertSim predictions. For ease of comparison, we highlight where the original BOLA1 and BBA lie. CausalSim confirms our suspicion; the curve for BOLA1 is strictly better than that of BBA. We can revise the hyperparameters in BOLA1 for an improved BOLA1 variant, henceforth called 'BOLA1-CausalSim'. We chose BOLA1-CausalSim, such that it would have better stall rate and marginally better SSIM compared to BBA.

Interestingly, ExpertSim predicts the complete opposite. It predicts that not only will BBA always improve on any BOLA1 variant in at least one metric, but also that any BOLA1 variant will stall more. This serves as a great opportunity to test CausalSim's edge compared to traditional (biased) trace-driven simulation, which is used in prior work [38,50,75]. The results of BOLA1-CausalSim's deployment can be seen in Figure 5. Considering confidence intervals, it is clear that it stalls less than BBA; in fact, BBA stalls 43% more than BOLA1-CausalSim on average. The confidence intervals for

[12] The data for this plot comes directly from Puffer [2, 3].

[13] We use a Gaussian Process prior with a Matern Kernel [54].

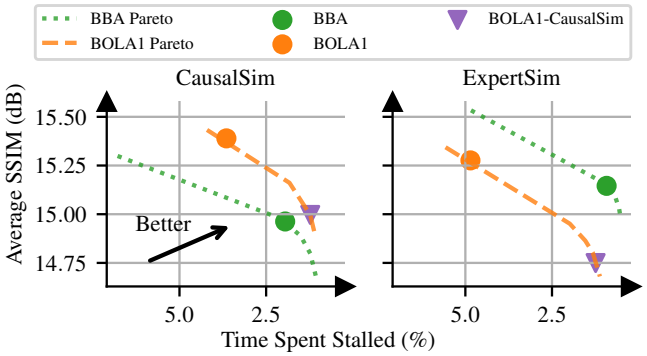


Figure 6: Pareto frontier curves for BOLA1 and BBA variants. **CausalSim correctly predicts BOLA1's potential**, while ExpertSim fails to do so.

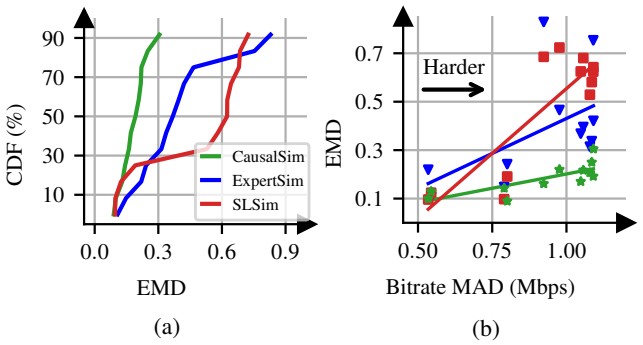


Figure 7: On average, CausalSim improves the EMD distance metric compared to ExpertSim and SLSim by 53% and 61% respectively. **(a)** Distribution of CausalSim, ExpertSim, and SLSim EMDs over all possible source/target choices. **(b)** Error (EMD) increases for baseline as simulation scenarios get harder, but CausalSim maintains good accuracy.

quality are wide and will need more data to be separable[14], but based on the ongoing trend, BOLA1-CausalSim will have similar quality compared to BBA.

Our goal was to show CausalSim's potential, and for that we targeted one of several plots on Puffer ('slow streams'). We could have chosen a different plot to optimize on, but it would not affect the takeaway. Note that our opportunities for deployment on Puffer are limited, as other researchers use Puffer as well; hence we only deployed one BOLA1 variant. Furthermore, we hoped to also compare CausalSim's prediction of stall rate and quality with the deployment results, but the client and network population has clearly changed; as shown in Figure 5, BBA achieves a different SSIM value for the two periods of time. Since CausalSim's predictions are based on data from the previous RCT, directly comparing the predicted values to results from the new RCT isn't meaningful. However, as our results show, the old RCT data allows us to compare different schemes. For example, CausalSim predicts BBA stalls 58% more than BOLA1-CausalSim on network distribution of the old RCT, which is reasonably close to the 43% observed in the new RCT (ignoring confidence intervals).

6.3 A Closer Look at Simulated Trajectories

For a deep dive in simulator accuracy, we focus on buffer occupancy level, a key indicator of ABR algorithm behavior. Ideally, we would like to compare simulated trajectories to ground truth. But this isn't possible using real trace data, since it requires us to have multiple traces of different policies running under the exact same underlying path conditions. To overcome this issue, we resort to distributional evaluation. Puffer data is collected in an RCT setting; hence the characteristics of network paths assigned to each policy is the same. If we accurately simulate the target policy on traces assigned to one of the source policies, the distribution of each variable (e.g. buffer level) must be similar in the simulated trajectory and ground truth trace assigned to the target policy. This motivates using distributional similarity as our performance metric.

To quantify the similarity of two distributions, we use the Earth Mover Distance (EMD) [62]. We can calculate EMD for one-dimensional distributions as $\text{EMD}(\mathcal{P}, Q) = \int_{-\infty}^{+\infty} |\mathcal{P}(x) - Q(x)| dx$, where $\mathcal{P}$ and Q are the Cumulative Distribution Function (CDF)s of p and q, respectively. A small EMD between two distributions implies that they are similar.

Figure 7a shows the CDF of the EMD (between actual and simulated buffer level distributions) for CausalSim and baselines, over all possible source/target policy pairs. EMD of CausalSim is smaller than EMD of baselines across almost all experiments. In terms of the average EMD across all experiments, CausalSim bests ExpertSim and SLSim by 53% and 61% respectively. Figure 2a visualized differences in buffer level distributions for the simulation scenario where BOLA2 and BBA are source and target policies, respectively. To observe buffer level distributions for all scenarios, refer to Figure 9.

In about 30% of cases, SLSim is slightly better than CausalSim. These cases are "easy" simulation scenarios where the source and target policies make similar actions (For more details see §B.3). In these cases, the EMD is low for both CausalSim and baseline simulators (< 0.15), and all perform well. For instance, Figure 9c (in the Appendix) shows source, target, and simulated buffer level distribution in an easy scenario, where BOLA2 and BOLA1 are the target and source policies respectively. In this example, all simulated distributions match the target distribution quite well.

Figure 7b shows where CausalSim most shines, i.e. hard simulation scenarios. The Y-axis is the error (EMD), and the X-axis is the mean absolute difference (MAD) between actions taken by the source policy and the target policy, in SLSim simulation. The larger the action difference, the harder the scenario (§B.3). As we move toward harder scenarios, the error increases

[14] Updated plots can be found on the 'Experimental Results' page of the Puffer website [1], under "Current experiment, full contiguous duration, slow streams only".

significantly for the baselines, while CausalSim is more robust.

6.3.1 Additional experiments

We perform further evaluations of CausalSim in the ABR environment. Due to space constraints, we summarize these results here and defer details to the appendix.

A more fine-grained evaluation. In the results above, we evaluated the performance of CausalSim and baselines using the distribution of buffer occupancy across the whole population. One way to further validate the results is to test whether they will hold on carefully partitioned sub-populations. In §B.4, we show that this is indeed the case when the sub-populations are partitioned according to the Min Round Trip Time (RTT), a network property that is independent of the selected ABR algorithm in Puffer.

Hyperparameters tuning. Counterfactual estimation (§3.2) is inherently an Out of Distribution (OOD) prediction task. Hence, typical supervised-learning hyper-parameter tuning methods do not work. In §B.5, we describe and evaluate CausalSim's hyper-parameter tuning procedure.

Ground truth evaluation. Real data never comes with ground truth counterfactual labels. As a result, we cannot evaluate CausalSim's simulations for each time step in real data, but we can do this in a reproducible synthetic environment. In §C.2, we evaluate CausalSim using ground truth counterfactual labels and show that it still outperforms baselines in the Mean Absolute Percentage Error (MAPE) metric.[15] Specfically, CausalSim achieves an MAPE of (∼5%), which is significantly lower than both ExpertSim's and SLSim's (∼10%).

6.4 A Second Example: Server Load Balancing

We now focus on simulating load balancing policies with heterogeneous servers, where defining an exogenous trace is not possible and therefore standard trace-driven simulation is not applicable. This example shows how CausalSim opens up new avenues in trace-driven simulation.

We use a synthetic environment which consists of $N = 8$ servers (and a queue for each) with different processing powers, a load balancer, and a series of jobs that need to be processed on these servers. Each job has a specific size which is unknown to the load balancer. Each server can process jobs at a specific rate $\{r_i\}_{i=1}^N$, which is also unknown to the load balancer. The load balancer receives jobs and must 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/r_{a_k}. If this job is not blocked by some other job being processed, its latency will equal 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 $S_k/r_{a_k}+T_k$.

[15] Let $\hat{\mathbf{p}} = \{\hat{p}_i\}_{i=1}^N$ and $\mathbf{p} = \{p_i\}_{i=1}^N$ denotes the vectors of predicted and ground truth quantity of interest, respectively. Then, MAPE is defined as $\text{MAPE}(\mathbf{p},\hat{\mathbf{p}}) = \frac{100}{N}\sum_{i=1}^N \frac{|\hat{p}_i - p_i|}{p_i}$.

We generate a collection of 5000 trajectories each with 1000 steps and use 16 policies in the load balancer. For a detailed explanation of the policies, job size generation process, and server processing rates, refer to §D.2.

6.4.1 Experiment setup

The aim of this experiment is to evaluate whether we can simulate new unseen server assignment policies in this environment, using traces collected with other policies. Recall that while we observe the processing time of each job, the actual size of the job is not observed, i.e., it acts as the latent factor in this problem. For all simulators, we assume access to $\mathcal{F}_{\text{system}}$ (the queue model) and focus on the more challenging task of learning $\mathcal{F}_{\text{trace}}$ and estimating the counterfactual traces $\hat{m}_i^t$ for $i \leq 5000$, and $t \leq 1000$. Algorithmically, this translates to enforcing consistency for the observed traces (m_t), rather than the observations (o_t) (see §5). The trace we collect is the processing time when using a *source* server assignment policy. To simulate a *target* server assignment policy, we need to estimate the processing time of a job on servers other than the one where its processing time was measured (without knowing either the job size or the server processing rates).

Standard trace-driven simulation assumes an exogenous trace (job processing time), but this is the same as assuming servers have equal processing rates. This contradicts the problem setup, and standard trace-driven simulation (analogous to ExpertSim in ABR) is not applicable to this problem. Thus, we compare CausalSim with *SLSim* simulations. SLSim (realized by an NN) takes as input the observed processing time and the target server, and its output is the processing time under the targeted server. However, the observed and target processing time are always the same in training data, and hence it is impossible for SLSim to learn the true dynamics (e.g., the server's underlying processing power). CausalSim sidesteps this problem by explicitly estimating latent factors. For details regarding the network architecture and training details for both SLSim and CausalSim, refer to Table 8 in the appendix.

Performance Metric. We compare CausalSim and SLSim with the underlying ground truth using the MAPE metric.

6.4.2 Can CausalSim Faithfully Simulate New Policies?

As is done in the ABR case studies, we train CausalSim and SLSim models based on a dataset generated using all policies except one, which will be the target policy. We use the same hyper-parameter tuning approaches explained in §B.5 for CausalSim and §B.6 for SLSim. We carry out this evaluation on eight target policies. We evaluate the performance for each pair of source-target policies, as was done in §6.1. In total, we have 120 different source/target policy pairs.

In Figure 8a and Figure 8b, we show the CDF of the MAPE of estimating the processing time and the latency, respectively, using both CausalSim and SLSim. As evident in these two figures, CausalSim's error is significantly lower than that of

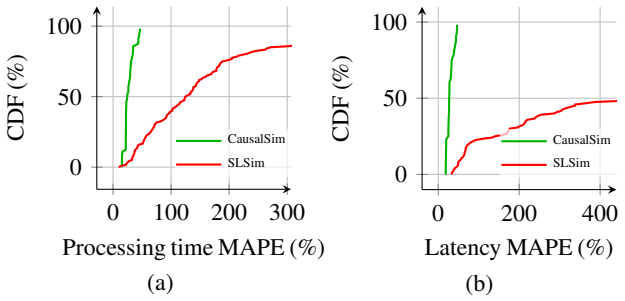


Figure 8: Distribution of CausalSim and SLSim MAPEs over all source target pairs.

SLSim for both the processing time and latency. In particular, the median MAPE when estimating processing time/latency is 24.4%/27.0% for CausalSim and 124.3%/467.8% for SLSim. For a complementary view, we compare the latent factors CausalSim extracts to the real latent job sizes and observe how closely they match, in §D.1 in the appendix.

7 Related-Work

Data-driven simulation. Traditional packet-level simulators [21, 31, 45] tend to sacrifice either scalability or accuracy when simulating large networks. MimicNet [77] and DeepQueueNet [73] use machine learning to improve simulation speed of datacenter networks. The aforementioned approaches are all full-system packet-level simulators, whereas CausalSim focuses on trace-driven simulation of a specific system component and must therefore deal with latent factors and biases present in trace data.

A very recent work, Veritas [17] (published on arXiv in Aug. 2022), models trace-driven simulation for ABR as a Hidden Markov Model (HMM) with a known emission process. This is equivalent to assuming that $\mathcal{F}_{\text{trace}}$ is known in our model (see Eq. (1)). Veritas uses the Viterbi algorithm to decode the latent factors, which are then used for counterfactual simulation. CausalSim solves a more general problem where $\mathcal{F}_{\text{trace}}$ is not known and must be learned. It therefore requires less knowledge of the system's latents and underlying dynamics to apply. On the other hand, CausalSim requires RCT data whereas Veritas does not. Comparing the fidelity of these approaches using real-world ABR data would be interesting future work (Veritas evaluates its method in a network emulator).

Panthon's calibrated emulators [72] model the end-to-end behaviour of a network path with a simple model including a handful of parameters, e.g., bottleneck link rate, constant propagation delay, etc., which are tuned to fit a collection of packet traces collected from this path using a variety of congestion control protocols. iBox [13] extends this approach by modeling cross-traffic. CausalSim does not assume any known model for the dynamics of the network. Furthermore, it has access to only a single trace from each network path.

Policy evaluation. Policy evaluation techniques such as Inverse Propensity Scoring [33] and Doubly Robust [15] aim to predict population-level performance statistics for a given intervention. WISE [67] builds a Causal Bayesian Network from the data that is able to answer interventional (what-if) queries about the future, but the method requires absence of latent confounding variables. Sage [25] uses a Causal Bayesian Network model with latent factors to diagnose performance issues in microservice applications. It answers what-if questions about how interventions like changing the resources allocated to a microservice impacts the end-to-end application latency. Trace-driven simulation is distinct from all these methods, in that it requires counterfactual predictions of how an intervention would have changed specific previously-measured trajectories rather than how it changes population-level statistics.[16]

8 Concluding Remarks

The exogenous trace assumption is central to traditional trace-driven simulation. CausalSim relaxes this key assumption, by modeling the intervention effect on the trace and learning to replay the trace in an unbiased manner. We showed how this improves the accuracy of trace-driven simulation using real-world ABR data, and how CausalSim provides insights for algorithm improvement that are in contrast with standard trace-driven simulators' predictions, which we validated in a real-world deployment. Furthermore, we showed how this expands the applicability of trace-driven simulation to problems where defining an exogenous trace is not possible by applying it to heterogeneous server load balancing. We believe CausalSim could be applied to many other system simulation tasks.

CausalSim opens up several interesting paths for future work. First, evaluating CausalSim in problems with a higher-dimensional latent factors would be interesting. Second, it is a natural next step to use CausalSim for more complex policy optimization methods, e.g., using reinforcement learning. Last, as discussed in §4.3, our theoretical analysis of CausalSim's approach, i.e. exploiting the policy invariance of latent factors distributions, is not tight, and improving it could potentially relax the assumptions of our analytical method.

9 Acknowledgement

We thank our shepherd Keith Winstein for in-depth suggestions, and our reviewers for insightful comments. We thank the Puffer team, specifically Emily Marx and Francis Y. Yan for providing us with the data we used in §6.1 and the algorithm deployment in §6.2. This work was supported by NSF grants 1751009 and 1955370, an award from the SystemsThatLearn@CSAIL program, and a gift from Intel as part of the MIT Data Systems and AI Lab (DSAIL). A. Alomar and D. Shah were supported in part by DSO-Singapore project, MIT-IBM project on Causal representation learning and NSF FODSI project.

[16] Appendix E provides a broader overview of the causal inference literature.

References

[1] Puffer: Experimental results. https://puffer.stanford.edu/results/. Accessed: 2023-2-22.

[2] Puffer: Total scheme statistics - decmeber 27th, 2022. https://storage.googleapis.com/puffer-data-release/2022-12-27T11_2022-12-28T11/duration_slow_scheme_stats_2022-12-27T11_2022-12-28T11.txt. Accessed: 2023-2-22.

[3] Puffer: Total scheme statistics - july 2nd, 2021. https://storage.googleapis.com/puffer-data-release/2021-06-01T11_2021-06-02T11/duration_slow_scheme_stats_2021-06-01T11_2021-06-02T11.txt. Accessed: 2023-2-22.

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

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

[6] 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.

[7] Anish Agarwal, Abdullah Alomar, and Devavrat Shah. On multivariate singular spectrum analysis. *arXiv e-prints*, pages arXiv–2006, 2020.

[8] Anish Agarwal, Munther A. Dahleh, Devavrat Shah, and Dennis Shen. Causal matrix completion. *ArXiv*, abs/2109.15154, 2021.

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

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

[11] 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.

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

[13] Sachin Ashok, Shubham Tiwari, Nagarajan Natarajan, Venkata N Padmanabhan, and Sundararajan Sellamanickam. Data-driven network path simulation with ibox. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–26, 2022.

[14] Susan Athey, Mohsen Bayati, Nikolay Doudchenko, Guido Imbens, and Khashayar Khosravi. Matrix completion methods for causal panel data models. *Journal of the American Statistical Association*, pages 1–15, 2021.

[15] 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.

[16] Vineet Bharti, Pankaj Kankar, Lokesh Setia, Gonca Gürsun, Anukool Lakhina, and Mark Crovella. Inferring invisible traffic. In *Proceedings of the 6th International COnference*, Co-NEXT '10, New York, NY, USA, 2010. Association for Computing Machinery.

[17] Chandan Bothra, Jianfei Gao, Sanjay Rao, and Bruno Ribeiro. Veritas: Answering causal queries from video streaming traces. arXiv/2208.12596, August 2022.

[18] Changxiao Cai, Gen Li, Yuejie Chi, H Vincent Poor, and Yuxin Chen. Subspace estimation from unbalanced and incomplete data matrices: $\ell_{2,\infty}$ statistical guarantees. *The Annals of Statistics*, 49(2):944–967, 2021.

[19] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009.

[20] Emmanuel J Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[21] 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.

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

[23] 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.

[24] 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.

[25] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: Practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 135–151, New York, NY, USA, 2021. Association for Computing Machinery.

[26] Silvia Gandy, Benjamin Recht, and Isao Yamada. Tensor completion and low-n-rank tensor recovery via convex optimization. *Inverse problems*, 27(2):025010, 2011.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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*, SIGCOMM '14, page 187–198, New York, NY, USA, 2014. Association for Computing Machinery.

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

[37] 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.

[38] 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.

[39] Maryia Kabanava, Holger Rauhut, and Ulrich Terstiege. On the minimal number of measurements in low-rank matrix recovery. In *2015 International Conference on Sampling Theory and Applications (SampTA)*, pages 382–386, 2015.

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

[41] Daniel Kressner, Michael Steinlechner, and Bart Vandereycken. Low-rank tensor completion by riemannian optimization. *BIT Numerical Mathematics*, 54(2):447–468, 2014.

[42] S Shunmuga 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.

[43] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review*, 34(4):219–230, 2004.

[44] Anukool Lakhina, Konstantina Papagiannaki, Mark Crovella, Christophe Diot, Eric D. Kolaczyk, and Nina Taft. Structural analysis of network traffic flows. *SIGMETRICS Perform. Eval. Rev.*, 32(1):61–72, jun 2004.

[45] 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.

[46] Yongjun Liao, Wei Du, Pierre Geurts, and Guy Leduc. Dmfsgd: A decentralized matrix factorization algorithm for network distance prediction. *IEEE/ACM Trans. Netw.*, 21(5):1511–1524, oct 2013.

[47] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[48] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. Tensor completion for estimating missing values in visual data. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):208–220, 2012.

[49] 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.

[50] 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.

[51] Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. Variance reduction for reinforcement learning in input-driven environments, 2018.

[52] Yun Mao, Lawrence K. Saul, and Jonathan M. Smith. Ides: An internet distance estimation service for large networks. *IEEE Journal on Selected Areas in Communications*, 24(12):2273–2284, 2006.

[53] Emily Marx, Francis Y. Yan, and Keith Winstein. Implementing bola-basic on puffer: Lessons for the use of ssim in abr logic, 2020.

[54] Bertil Matérn. *Spatial variation*, volume 36. Springer Science & Business Media, 2013.

[55] 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.

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

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

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

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

[60] Matthew Roughan, Yin Zhang, Walter Willinger, and Lili Qiu. Spatio-temporal compressive sensing and internet traffic matrices (extended version). *IEEE/ACM Transactions on Networking*, 20(3):662–676, 2012.

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

[62] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE, 1998.

[63] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, 2020.

[64] P. 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*, NetAI '20, page 42–47, New York, NY, USA, 2020. Association for Computing Machinery.

[65] 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.

[66] Liying Tang and Mark Crovella. Virtual landmarks for the internet. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, page 143–152, New York, NY, USA, 2003. Association for Computing Machinery.

[67] 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.

[68] 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.

[69] Madeleine Udell and Alex Townsend. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 1(1):144–160, 2019.

[70] Zhiqiang Xu. The minimal measurement number for low-rank matrix recovery. *Applied and Computational Harmonic Analysis*, 44(2):497–508, 2018.

[71] 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.

[72] 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, 2018.

[73] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. Deepqueuenet: Towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 441–457, New York, NY, USA, 2022. Association for Computing Machinery.

[74] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. Menet: Towards effective adversarial robustness with matrix estimation. *arXiv preprint arXiv:1905.11971*, 2019.

[75] 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.

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

[77] 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.

[78] Zemin Zhang and Shuchin Aeron. Exact tensor completion using t-svd. *IEEE Transactions on Signal Processing*, 65(6):1511–1526, 2016.

Appendix A Tensor Completion with policy invariance

Here, we discuss a more generic version of the problem considered in §4.2 from the lens of tensor completion. Specifically, in §4 we considered the simplified setting where the trace was considered to be one-dimensional. Here, we shall consider higher dimensional traces. This, naturally suggests using the lens of Tensor instead of Matrix completion. We will also discuss how higher dimensional trace can enable recovery of more complex system dynamics or models compared to the simple solution we discussed in §4 for rank 1 setup.

Potential Outcomes Tensor. As considered in §4 let all possible actions be denoted as $[A] = \{1, \dots, A\}$ for some $A \geq 2$. Let the trace be of D dimension. As before, we have N trajectories of interest with trajectory $i \in [N]$ being of length $H_i \geq 1$ time steps. As before, let $U = \sum_{i=1}^{N} H_i$.

Consider an order-3 tensor M of dimension $A \times U \times D$, where $M = [m_{\alpha\beta\gamma} : \alpha \in [A], \beta \in [U], \gamma \in [D]]$ with $m_{\alpha\beta\gamma}$ corresponds to the γth co-ordinate of the D-dimensional trace corresponding to action $a_t = \alpha \in [A]$ when latent factor is $u_{i,t}$ with β corresponding to enumeration of (i,t) for some $i \in [N]$ and $t \leq H_i$. Recall that, as explained in Section 4, all possible $(i,t) : t \leq H_i, i \in [N]$ are mapped to an integer in $[U]$. We call this tensor M as the Potential Outcomes Tensor.

Indeed, if we know M completely, then we can answer the task of simulation or counterfactual estimation well since we will be able to estimate the mediator for each trajectory under a given possible sequence of counterfactual actions, and subsequently estimate the counterfactual observation (assuming we could learn the $\mathcal{F}_{\text{systems}}$).

We shall assume that there are $P \geq 1$ policies under which these traces where observed. In particular, each trajectory was observed under one of these P policies and the assignment of policy to the trajectory was done uniformly at random. Define $\Pi_p \subset [U]$ as collection of indices corresponding to trajectories $i \in [N]$ and their times $t \leq H_i$ where trajectory i was assigned policy p for $p \in [P]$. Let $U_p = |\Pi_p|$.

Tensor factorization, low CP-rank. The tensor M admits (not necessarily unique) factorization of the form: for any $\alpha \in [A], \beta \in [U], \gamma \in [D]$

$$m_{\alpha\beta\gamma} = \sum_{\ell=1}^{r} x_{\alpha\ell} y_{\beta\ell} z_{\gamma\ell}, \tag{8}$$

for some $r \geq 1$. For any tensor, such a factorization exits with r at most $\text{poly}(A,U,D)$.

*Assumption 1 (**low-rank factorization**).* We shall make an assumption that r is *small*, i.e. does not scale with A,U,D and specifically a small constant.

*Assumption 2 (**sufficient measurements**).* We shall assume that number of measurements per instance, D, is at least as large as the underlying rank r of the tensor M, i.e. $D \geq r$.

Distributional invariance and RCT. As before, we shall assume that the distribution of latent factors is the same across different policies due to random assignment of policies to trajectories in the setup of RCT. In the context of the tensor M, this corresponds to the distribution invariance of factors $y_{\beta\cdot} \in \mathbb{R}^r$ over $\beta \in \Pi_p$ for any $p \in [P]$. Concretely, for any $p \neq p' \in [P]$ and $\ell \in [r]$, we have

$$\frac{1}{U_p} \sum_{\beta \in \Pi_p} y_{\beta\ell} \approx \frac{1}{U_{p'}} \sum_{\beta' \in \Pi_{p'}} y_{\beta'\ell}. \tag{9}$$

More generally, any finite moment (not just first moment or average) of latent factors should be empirically invariant across policies. As in §4, we would like to utilize property (9) to estimate the tensor M.

A Simple Estimation Method and When It Works. We describe a simple method that can recover entire tensor as long as rank $r \leq D$. For simplicity, we shall assume $r = D$ (the largest possible rank for which method will work). By (8), for a given fixed $\alpha \in [A]$ and across $\beta \in [U], \gamma \in [D]$,

$$m_{\alpha\beta\gamma} = \sum_{\ell=1}^{r} y_{\beta\ell} \tilde{z}^{\alpha}_{\gamma\ell}, \tag{10}$$

where $\tilde{z}^{\alpha}_{\gamma\ell} = x_{\alpha\ell} z_{\gamma\ell}$. Since $D = r$, the matrix $\tilde{Z}^{\alpha} = [\tilde{z}^{\alpha}_{\gamma\ell} : \gamma \in [D], \ell \in [r]]$ is a square matrix. With this notation, we have that for any fixed $\alpha \in [A]$, the matrix $M^{\alpha} = [m_{\alpha\beta\gamma} : \beta \in [U], \gamma \in [D]] \in \mathbb{R}^{U \times D}$ (or $\mathbb{R}^{U \times r}$ since $r = D$) can be represented as

$$M^{\alpha} = Y \tilde{Z}^{\alpha,T}, \tag{11}$$

where $Y = [y_{\beta\ell} : \beta \in [U], \ell \in [r]] \in \mathbb{R}^{U \times r}$.

*Assumption 3 (**invertibility**).* We shall assume that the $D \times D$ (i.e. $r \times r$) square matrices $\tilde{Z}^{\alpha}$ for each $\alpha \in [A]$ are full rank and hence invertible.

The *Assumption 3* implies that $Y = M^{\alpha} \left(\tilde{Z}^{\alpha,T}\right)^{-1}$ for all $\alpha \in [A]$.

For policy $p \in [P]$, indices $\beta \in \Pi_p$ are relevant. For a given $\beta \in \Pi_p$, if the policy p utilized action $\alpha \in [A]$, $m_{\alpha\beta\cdot} \in \mathbb{R}^D$ is observed. To that end, let $\Pi_{p,\alpha} = \{\beta \in \Pi_p : \text{policy utilized action } \alpha\}$. Let $U_{p,\alpha} = |\Pi_{p,\alpha}|$ for any $\alpha \in [A]$. Then, define $Y^{p,\alpha} = [y_{\beta\ell} : \beta \in \Pi_{p,\alpha}, \ell \in [r]] \in \mathbb{R}^{U_{p,\alpha} \times r}$, $M^{\alpha,p} = [m_{\alpha\beta\gamma} : \beta \in \Pi_{p,\alpha}, \gamma \in [D]]$. Then we have $Y^{p,\alpha} = M^{\alpha,p} \left(\tilde{Z}^{\alpha,T}\right)^{-1}$.

Therefore, for any $\ell \in [r = D]$,

$$\begin{aligned} \sum_{\beta \in \Pi_{p,\alpha}} y_{\beta\ell} &= \mathbf{1}^{p,\alpha,T} Y^{p,\alpha} \mathbf{e}_{\ell} \\ &= \mathbf{e}_{\ell}^{T} Y^{p,\alpha,T} \mathbf{1}^{p,\alpha} \\ &= \mathbf{e}_{\ell}^{T} \left(\tilde{Z}^{\alpha}\right)^{-1} M^{\alpha,p,T} \mathbf{1}^{p,\alpha}, \end{aligned} \tag{12}$$

where $\mathbf{1}^{p,\alpha} \in \mathbb{R}^{U_{p,\alpha}}$ is vector of all 1s, and $\mathbf{e}_{\ell} \in \mathbb{R}^r$ be vector with all entries 0 but the $\ell \in [r]$th co-ordinate 1 .

Then, for any $\ell \in [r]$ and $p \in [P]$,

$$\begin{aligned}\frac{1}{U_p}\sum_{\beta\in\Pi_p} y_{\beta\ell} &= \frac{1}{U_p}\sum_{\alpha\in[A]}\sum_{\beta\in\Pi_{p,\alpha}} y_{\beta\ell} \\ &= \frac{1}{U_p}\sum_{\alpha\in[A]} \mathbf{e}_\ell^T \left(\tilde{Z}^\alpha\right)^{-1} M^{\alpha,p,T}\mathbf{1}^{p,\alpha} \\ &= \sum_{\alpha\in[A]} \mathbf{e}_\ell^T \left(\tilde{Z}^\alpha\right)^{-1}\left(\frac{1}{U_p}M^{\alpha,p,T}\mathbf{1}^{p,\alpha}\right) \\ &= \sum_{\alpha\in[A]} \mathbf{e}_\ell^T \left(\tilde{Z}^\alpha\right)^{-1}\mathcal{M}^{\alpha,p}, \end{aligned} \tag{13}$$

where $\mathcal{M}^{\alpha,p} = \frac{1}{U_p}M^{\alpha,p,T}\mathbf{1}^{p,\alpha} \in \mathbb{R}^{r,1}$ is an observed quantity, while $\tilde{Z}^{\alpha,T}$ is unknown. Using (13) and (9), we obtain that for any $\ell \in [r]$ and $p \neq p' \in [P]$,

$$\sum_{\alpha\in[A]} \mathbf{e}_\ell^T \left(\tilde{Z}^\alpha\right)^{-1}\mathcal{M}^{\alpha,p} \approx \sum_{\alpha\in[A]} \mathbf{e}_\ell^T \left(\tilde{Z}^\alpha\right)^{-1}\mathcal{M}^{\alpha,p'}. \tag{14}$$

Let $\tilde{z}^{\alpha,\ell} = \mathbf{e}_\ell^T \left(\tilde{Z}^\alpha\right)^{-1} \in \mathbb{R}^{1,r}$ be the ℓth row the of $r\times r$ matrix $\left(\tilde{Z}^\alpha\right)^{-1}$. Then (14) implies that for any $\ell \in [r]$ and $p \neq p' \in [P]$,

$$\sum_{\alpha\in[A]} \tilde{z}^{\alpha,\ell}(\mathcal{M}^{\alpha,p} - \mathcal{M}^{\alpha,p'}) \approx 0. \tag{15}$$

Which can be written in matrix form as

$$\begin{bmatrix}\tilde{z}^{1,\ell} & \tilde{z}^{2,\ell} & \dots & \tilde{z}^{A,\ell}\end{bmatrix}\begin{bmatrix}\mathcal{M}^{1,p}-\mathcal{M}^{1,p'} \\ \mathcal{M}^{2,p}-\mathcal{M}^{2,p'} \\ \vdots \\ \mathcal{M}^{A,p}-\mathcal{M}^{A,p'}\end{bmatrix} = 0 \tag{16}$$

By noting that that this hold for all $\ell \in [r]$, and recalling that $\tilde{z}^{\alpha,\ell}$ is the ℓ-th row the of the $r\times r$ matrix $\left(\tilde{Z}^\alpha\right)^{-1}$, we get,

$$\begin{bmatrix}\left(\tilde{Z}^1\right)^{-1} & \left(\tilde{Z}^2\right)^{-1} & \dots & \left(\tilde{Z}^A\right)^{-1}\end{bmatrix}\begin{bmatrix}\mathcal{M}^{1,p}-\mathcal{M}^{1,p'} \\ \mathcal{M}^{2,p}-\mathcal{M}^{2,p'} \\ \vdots \\ \mathcal{M}^{A,p}-\mathcal{M}^{A,p'}\end{bmatrix} = \mathbf{0}, \tag{17}$$

where $\mathbf{0}$ is a vector of zeros of size r. Note that the above is a system of r linear equations, with Ar^2 unknowns (recall that the $r\times r$ matrices $\left(\tilde{Z}^\alpha\right)^{-1}$ are unknown for $\alpha \in [A]$). Let $\mathbf{Z} \in \mathbb{R}^{r\times Ar}$ and $\mathbf{v}^{p,p'} \in \mathbb{R}^{Ar}$ denote the first and second matrix in the left hand side, respectively, then (17) can be re-written as,

$$\mathbf{Z}\mathbf{v}^{p,p'} \approx \mathbf{0}. \tag{18}$$

By definition, $\mathbf{v}^{p,p'}$ is observed quantity for each $p \neq p' \in [P]$. Now if we consider $P-1$ equations produced by considering pair of policies $(1,2),(1,3),\dots,(1,P)$ in (18), by design they are non-redundant linear equations. Let matrix $\mathbf{V} \in \mathbb{R}^{Ar\times P-1}$ be formed by stacking $\mathbf{v}^{1,2},\dots,\mathbf{v}^{1,P}$ column-wise.

Furthermore, let us define $\mathbf{s}^p \in \mathbb{R}^{Ar}$ as $[\mathcal{M}^{1,p},\cdots,\mathcal{M}^{A,p}]^\intercal$. Define $\mathbf{S} \in \mathbb{R}^{Ar\times P}$ by stacking $\mathbf{s}^1,\cdots,\mathbf{s}^P$ column-wise.

*Assumption 4 (**Sufficient, Diverse Policies**).* Let $P \geq Ar$ and the rank of $\mathbf{S} = Ar$.

Note that we can derive $\mathbf{V}$ from $\mathbf{S}$ by subtracting the first column from all other columns, and removing the first column. Thus, Under Assumption 4, the +rank of $\mathbf{V}$ is at least $Ar-1$. Further, given Assumption 3 which excludes the scenario $\mathbf{Z}=0$, it follows that the rank of $\mathbf{V}$ is $Ar-1$. As rank of $\mathbf{V}$ is $Ar-1$, we can uniquely (up to scaling) recover $\mathbf{Z}$ by solving for system of linear equation $\mathbf{ZV}=\mathbf{0}$ as the null space of $\mathbf{V}$ is of dimension 1.

Once we know $\mathbf{z}$, i.e. by undoing flattening, we obtain $\left(\tilde{Z}^{\alpha,T}\right)^{-1}$ for each $\alpha \in [A]$. Since for each policy $p \in [P]$ and $\alpha \in [A]$, $Y^{p,\alpha} = M^{\alpha,p}\left(\tilde{Z}^{\alpha,T}\right)^{-1}$ and we observe $M^{\alpha,p}$, we can recover $Y^{p,\alpha}$ and hence subsequently $Y \in \mathbb{R}^{U\times r}$.

By (11), we can now recover slice of tensor M, the M^α for each $\alpha \in [A]$, and hence we can recover entire tensor M as desired.

Interpretation of Assumption 4. Consider β^{th} Column of the matrix $\mathbf{S}$, i.e., $\left[\mathbb{E}[m^\intercal|i=1,\pi_\beta]\mathbb{P}(i=1|\pi_\beta),\cdots,\mathbb{E}[m^\intercal|i=A,\pi_\beta]\mathbb{P}(i=A|\pi_\beta)\right]^\intercal$ where i denotes the action index and β the policy index. This column is a vector of statistics associated with traces collected using policy β. Each element in this vector consists of two components: the first component is the conditional mean of the trace given a specific action, and the second element is the probability of taking this action. We interpret linear independence of each of these components for different policy vectors as policy diversity. For instance, think of the second component which captures probability vectors of different actions for each policy. Its linear independence across different policies roughly means that each policy should assign new probability vectors to different actions, and not a probability vector similar (linearly dependent) to that of previous policies. Also note that this assumption is not satisfied if an action is not taken by any of the policies which makes all elements of the corresponding row equal to zero.

Appendix B Real-world ABR

B.1 Comprehensive results

In Figure 7a, we presented a concise view of simulator fidelity, for an internal variable in ABR sessions called buffer occupancy level. Specifically, we considered the simulation of a target policy, given trajectories collected using a different source policy. We measured the error between buffer simulations and ground truth through EMD, a similarity index for distributions. For a complementary view, we provide the full distributions in Figure 9, for all simulators and ground truth for target and source policies. Below each plot, we also report the EMD of CausalSim predictions.

B.2 Policy Discriminator and Latent Invariance

The policy discriminator ($\mathcal{W}_\gamma$ in Figure 3) described in §5 has the goal of predicting the source policy, given a latent factor generated by the latent factor extractor ($\mathcal{E}_\theta$ in Figure 3). Since our data is collected with an RCT, the true latent factor distribution should be indifferent to the source policy. Therefore, if the latent factor extractor generates the ground truth latent factors, the policy discriminator should not be able to predict the source policy accurately. In fact, even the optimal policy discriminator outputs the population share of each source policy (e.g. what fraction of the data comes from BBA) in the training data [28]. To assess this statement, we present the confusion matrix and population share of source data, for three left-out policies in Table 1. Each row corresponds to one source policy, and each column corresponds to the policy discriminator's prediction of the source policy. We observe that predictions do not change noticeably with different source policies, and that they closely match the population share for each left-out policy. This demonstrates that the extracted latent features were indeed invariant to the source policy.

B.3 What makes a simulation scenario easy/hard?

In §6.3, we compared the accuracy of CausalSim, ExpertSim and SLSim, in a simulation task on real ABR data. We observed that in about 30% of scenarios, which we call *easy* scenarios, all simulators perform well. However, in about 70% of the source/target scenarios, which we call *hard* simulation scenarios, baseline predictions are highly biased towards the source distributions. In these hard scenarios, CausalSim is able to de-bias the trajectories and its predictions match the target distribution well, as observable in Figure 9.

So it is natural to wonder what makes a simulation scenario easy/hard? An easy simulation scenario happens when source and target policies take similar actions. Similar action means that the factual achieved throughput (of the source policy) is similar to the counterfactual achieved throughput (of the target policy). This is what both ExpertSim (explicitly) and SLSim (implicitly) assume for doing simulation. Making this assumption is the core reason their simulations are biased in hard cases, where source and target policies take different actions, as we discussed in detail in §2.2.3.

Figure 10 validates our reasoning for what makes a simulation scenario difficult. The X axis shows the Mean Absolute Difference (MAD) between source and simulation actions (bitrates) when simulating with SLSim in a specific

	Prediction			
Source Policy	BOLA2	BOLA1	Fugu-CL	Fugu-2019
BOLA2	22.44%	22.58%	26.99%	27.99%
BOLA1	22.43%	22.58%	26.99%	27.99%
Fugu-CL	22.44%	22.58%	26.99%	27.99%
Fugu-2019	22.44%	22.58%	26.99%	28.00%
	Source Policy			
	BOLA2	BOLA1	Fugu-CL	Fugu-2019
Population	22.45%	22.50%	27.11%	27.94%

(a) Left-out policy is BBA

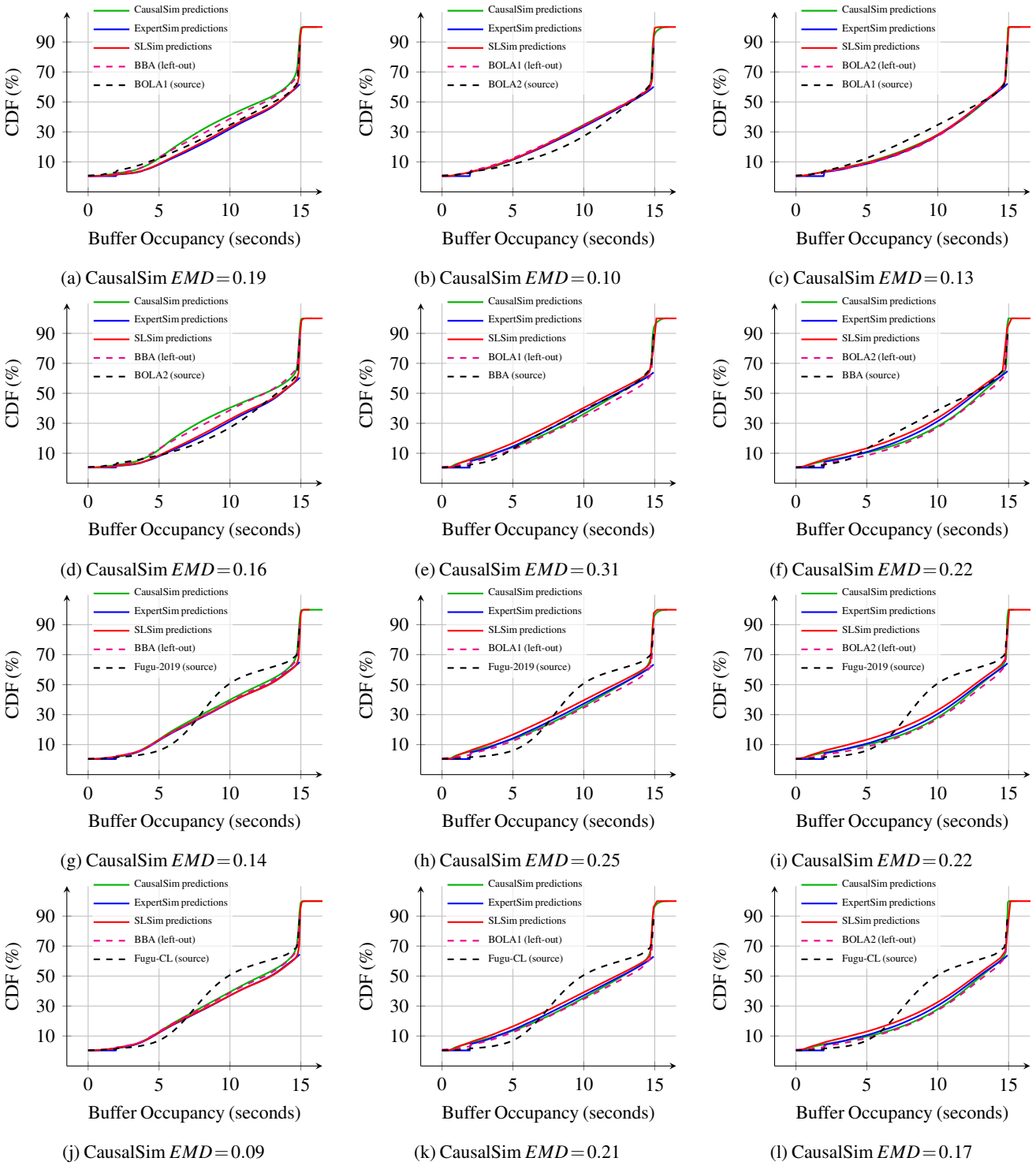
	Predictions			
Source Policy	BOLA2	Fugu-CL	Fugu-2019	BBA
BOLA2	21.34%	26.04%	26.75%	25.87%
Fugu-CL	21.33%	26.05%	26.75%	25.87%
Fugu-2019	21.33%	26.04%	26.77%	25.86%
BBA	21.33%	26.04%	26.76%	25.87%
	Source Policy			
	BOLA2	Fugu-CL	Fugu-2019	BBA
Population	21.48%	25.94%	26.74%	25.84%

(b) Left-out policy is BOLA1

	Predictions			
Source Policy	BOLA1	Fugu-CL	Fugu-2019	BBA
BOLA1	21.46%	26.00%	26.76%	25.78%
Fugu-CL	21.45%	26.01%	26.77%	25.76%
Fugu-2019	21.45%	26.00%	26.79%	25.76%
BBA	21.45%	25.99%	26.76%	25.80%
	Source Policy			
	BOLA1	Fugu-CL	Fugu-2019	BBA
Population	21.52%	25.93%	26.72%	25.83%

(c) Left-out policy is BOLA2

Table 1: Confusion matrix and population statistics for the policy discriminator with three left out policies.



(a) CausalSim $EMD = 0.19$ (b) CausalSim $EMD = 0.10$ (c) CausalSim $EMD = 0.13$

(d) CausalSim $EMD = 0.16$ (e) CausalSim $EMD = 0.31$ (f) CausalSim $EMD = 0.22$

(g) CausalSim $EMD = 0.14$ (h) CausalSim $EMD = 0.25$ (i) CausalSim $EMD = 0.22$

(j) CausalSim $EMD = 0.09$ (k) CausalSim $EMD = 0.21$ (l) CausalSim $EMD = 0.17$

Figure 9: Buffer level distribution of source, target, CausalSim predictions, and baseline predictions across all source/target scenarios.

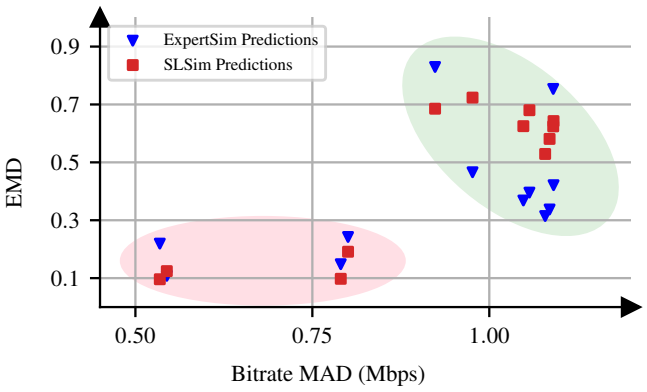


Figure 10: Simulation difficulty is related to how different counterfactual actions are from factual ones. This figure shows scatterplot of EMD versus mean absolute bitrate difference, for ExpertSim and SLSim, over all possible source left-out pairs. The pink cluster signifies the 'easy' scenarios and the green cluster signifies 'hard' ones.

source/target scenario. Y axis shows EMD (Our performance metric for simulation, smaller is better) of both baselines in that specific scenario.

Two main cluster of points are clearly visible in this figure. The pink cluster on the bottom left corresponds to easy simulations. It includes all source/target simulation scenarios where baselines perform well (bottom), and at the same time, source and target actions are quite similar (left).

The green cluster at the top right corresponds to the hard simulations. It includes all source/target simulation scenarios where baselines fail to perform an unbiased simulation (top), and at the same time, source and target actions are quite different (right).

B.4 A More Fine-grained Evaluation

Ideally, we would like to evaluate CausalSim's simulation to ground truth on a step-by-step basis for a given trajectory. But as discussed in §6.3, 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 steps in the observational data, which is referred to as the fundamental problem of Causal Inference [32]. This is the reason we evaluated predictions on a distributional level.

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. Min RTT is an inherent property of a network path[17], and we would expect Min RTT distribution to be the same for users assigned to different ABR policies.

We use the MinRTT to create the following four sub-populations:

1. Sub1: users with Min RTT $< 35^{ms}$
2. Sub2: users with $35^{ms} \leq$ Min RTT $< 70^{ms}$
3. Sub3: users with $70^{ms} \leq$ Min RTT $< 100^{ms}$
4. Sub4: users with $100^{ms} \leq$ 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 11a shows the CDF of CausalSim's EMD when simulating the left-out ABR algorithm over each of the above sub-populations. We can see that CausalSim maintains a superior EMD CDF compared to ExpertSim and SLSim, and remains accurate across different sub-populations. This further suggests that even at surgically small subpopulations, CausalSim maintains accuracy, and does not overfit to the whole distribution.

B.5 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 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 do not use 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 eleven different CausalSim models with different choices of κ (defined in Equation (7)). We consider two metrics: *(i) Test EMD*, defined as the average EMD when simulating the left-out ABR algorithm with trajectories in the training dataset. This is our main performance objective. *(ii) Validation EMD*, defined as the average EMD when

[17] This is true to a first order approximation, if we ignore the possibility that a video streaming session drives up queueing delays throughout the course of a video, thereby inflating the observed Min RTT.

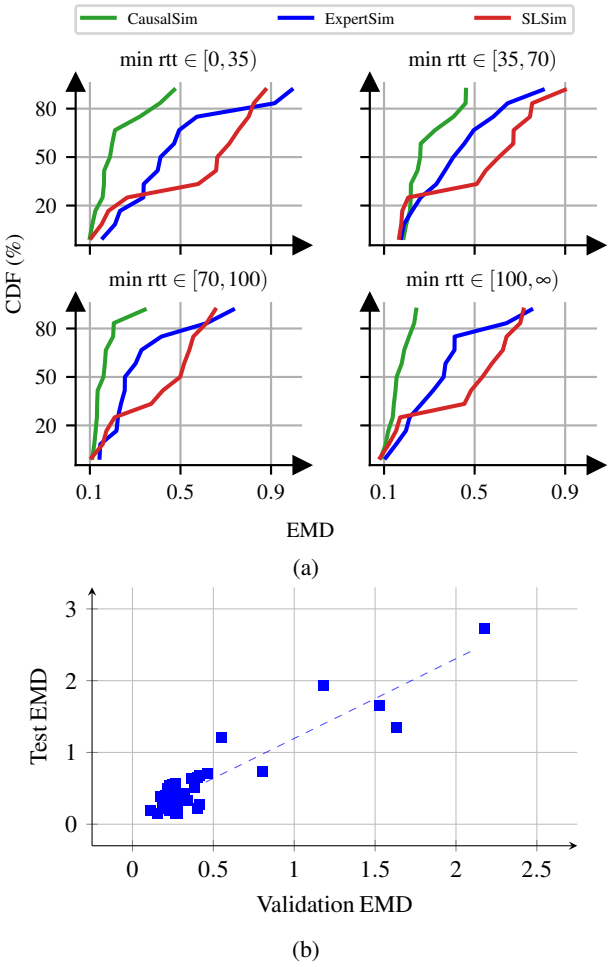


Figure 11: **(a)** Comparing the distribution of CausalSim EMDs with ExpertSim and SLSim over different sub-populations. **(b)** Validation EMD and test EMD are highly correlated. This justifies our hyper-parameter tuning strategy.

simulating ABR algorithms in the training dataset with trajectories in the training data that were collected with other ABR algorithms. This is our proxy objective for hyper-parameter tuning.

For each model (33 in all: 3 datasets, 11 example hyper-parameters), we calculate both Test EMD and Validation EMD, which results in one (Validation EMD, Test EMD) point in Figure 11b. The Pearson Correlation Coefficient (PCC) between Valid EMD and Test EMD is 0.92, 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 Validation EMD.

B.6 How to Tune SLSim's Hyper-parameters?

SLSim takes as input the current buffer value, selected chunk size and observed throughput, and similar to CausalSim, predicts the next buffer $\hat{b}_{t+1}$ and download time $\hat{d}_t$. We add two knobs to tune while training SLSim: **(1)** The loss function $\mathcal{L}_\xi(\cdot,\cdot)$ used to steer the NN output to the ground truth output, and **(2)** The relative weighting of the loss function for download time with respect to that of the buffer occupancy, η. Concretely, we use the following total loss:

$$\mathcal{L}_{\text{slsim}} = \mathbb{E}_B\left[\frac{1}{\eta+1}.\mathcal{L}_\xi(\hat{b}_{t+1},b_{t+1}) + \frac{\eta}{\eta+1}.\mathcal{L}_\xi(\hat{d}_t,d_t)\right] \quad (19)$$

where the expectation is over the a sampled minibatch B from dataset D, and b_{t+1} and d_t denote the ground truth values for next buffer level and chunk download time. Table 3 lists the loss functions and η values considered.

To tune these values, we use ground truth data from all policies except a left out policy. We then proceed with the proxy tuning objective used in §B.5, i.e. we look for the configuration with the highest accuracy at simulating algorithms in the training data using trajectories of other algorithms in the training data. We then use the resulting configuration (and model) to simulate the left-out policy on the training data.

From the perspective of tuning, this methodology puts SLSim on equal ground with respect to CausalSim, and makes for a fair comparison. Note that we do not tune loss function type or η with CausalSim due to limited computational resources, but tuning those as well could potentially improve CausalSim's accuracy.

B.7 Simulation Accuracy: Continued

In §6.1.1, we stated that ExpertSim and SLSim predictions are significantly affected by the source data they are simulating on, and demonstrated the effect of source policies on BOLA1 predictions in Figure 4b. Here, we demonstrate the same figure for BBA in Figure 12a and BOLA2 in Figure 12b. CausalSim is designed to remove the bias of the algorithm used for collecting source data when simulating a target policy and its predictions remains unaffected by the performance of that source policy. ExpertSim and SLSim however, due to the violation of the exogenous trace assumption, will predict different metrics when using different source traces.

B.8 Dataset & Algorithms

Our trajectories in the real-world (Puffer) data come from 'slow streams' in the time span of July 27, 2020 until June 2, 2021. In this period of time, 5 ABR algorithms appear consistently and are listed in Table 2. Each trajectory is an active client session streaming a live TV channel. We follow Puffer's definition of

Policies	Hyperparameter	Value	Used as source	Used as left out
BBA	Cushion	3 (as used in puffer)	✓	✓
	Reservoir	10.5 (as used in puffer)		
BOLA-BASIC v1	V	0.67 (As computed in puffer)	✓	✓
	γ	-0.43 (As computed in puffer)		
	Utility function	$\log_{10}(1-ssim)$ (As used in puffer)		
	Minimum utility	0 dB (As used in puffer)		
	Maximum utility	60 dB (As used in puffer)		
BOLA-BASIC v2	V	51.4 (As computed in puffer)	✓	✓
	γ	-0.43 (As computed in puffer)		
	Utility function	*ssim* (As used in puffer)		
	Minimum utility	0 (As used in puffer)		
	Maximum utility	1 (As used in puffer)		
Fugu-CL	-	-	✓	×
Fugu-2019	-	-	✓	×

Table 2: ABR algorithms used in the real-world dataset and experiments

'slow streams'; streams with TCP delivery rates below 6 Mbps. We use 'slow streams' data, since the highest quality chunks rarely surpass $6-7$ Mbps, and paths with higher bandwidth will always stream the highest quality chunks under all policies. Puffer uses the same reasoning and evaluates algorithms at two population levels; 'slow streams' and 'all streams'.

In aggregating 'slow stream' logs, we met several difficulties that we outline here for reproducibility. Data without these difficulties would potentially improve CausalSim's accuracy. Note that this does not affect Figure 5, as the data for that figure is reported directly on Puffer [2, 3].

Puffer logs are reported as three separate event groups; 1) 'video_sent': the first packet of a chunk is sent, 2) 'video_acked': The last packet of a chunk is acknowledged, 3) 'client': The client sent a message. Stall rate is computed using the 'client' logs and quality is computed using the 'video_sent' logs.

1. To compute download time, we have to merge 'video_sent' and 'video_acked', and ensure that merged logs are consecutive in timestamps, i.e. no chunk is missing in between two other chunks. However, in the current data this removes all chunks that have been sent but not acknowledged, usually the last chunk. Puffer uses these chunks in measuring quality level, but we can't. This did not have any measurable impact, however.

2. To compute stall rate, both total stall time and total watch time are computed with 'client' logs. For this, the latest report that obeys a set of rules is used. We, however, have to compute stall time and watch time using our merged logs (merged logs are also what we get out of simulation). This would be easy on the original data, if 'client' logs and 'video_sent' were in sync, but they are not; whenever a rebuffering is reported by the client, 'client' log is updated but 'video_sent' is updated in the next few chunks. To circumvent this, we recompute rebuffering as $t_r = max(0, t_d - b)$, where t_r is rebuffering, b is buffer occupancy and t_d is download time. This formula is off by half of an RTT, and empirically inflates stall rates by $1.26-1.31x$, for all policies. In the absence of synchronized data, this is the best we can recover, but it does not affect the comparison among policies. Hence, we believe simulating with this data should lead to similar trends as with clean unperturbed data.

3. We cannot calculate watch time as Puffer does, since we have to use the merged log. We tried several simple formulas that should calculate watch time, but oddly most turn out to be inaccurate. One reason is that in some streams, buffer playback rate is not 1, i.e. one second of buffer is not depleted per second. These streams are likely due to browser tabs put in background, and throttled by the browser threading system. As a workaround, we use the original watch time minus the original stall time that Puffer computed for a stream, and offset it by the total stall time in the simulation.

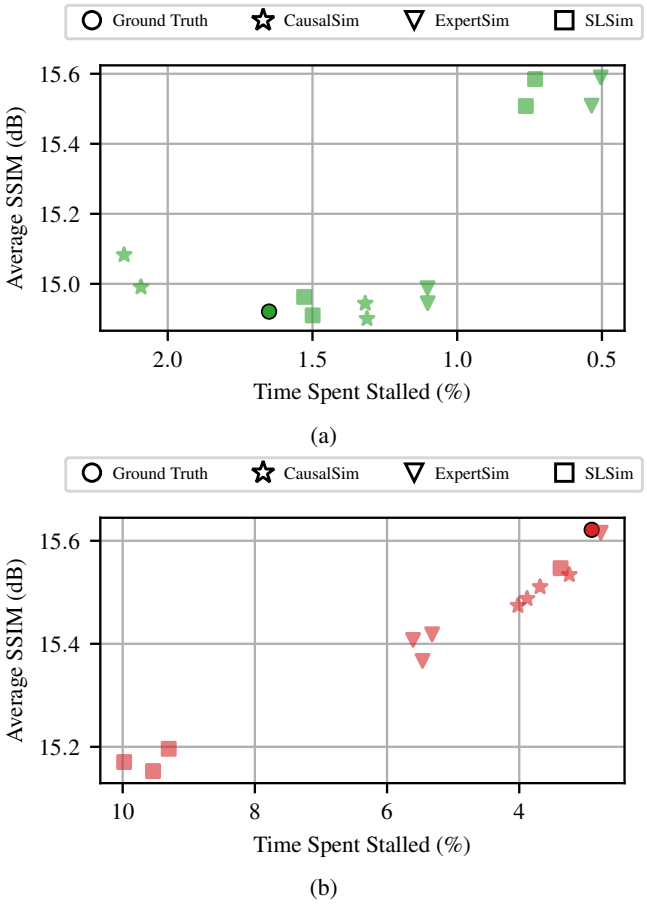


Figure 12: Predictions for **(a)** BBA and **(b)** BOLA2, separated by the ABR algorithm source data was collected with. Each point indicates a specific **source** ABR algorithm.

4. At each step, the buffer should not increase by more than a single chunk, 2.002 seconds, but it does (sometimes by as much as 14 seconds). We filter such data out.

5. When we are about to send a chunk, our last reported buffer value must never dip below 2.002 (except in the beginning). When buffer is below 15 seconds, the next chunk must be sent immediately after the last one. If rebuffering occurs, the next buffer value will be exactly 2.002 and if it doesn't, it will be larger than 2.002. We frequently (more than one million instances) observe buffer values below 2.002. We do not filter them out, as this would invalidate most logs.

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. To ensure our implementations are correct, we attempt to reconstruct the choices made at runtime by each policy, and compare them to the logged choices. We expect our reproduction to match 100% when our implementation is faithful and logs match runtime inputs. For the logs in July 27th, 2020, we observe 100% matching for BOLA1 and BOLA2 and 99.993% for BBA. For the latter, there are rare cases where two encodings are seemingly equal in SSIM up to the 6 logged decimal places, but were likely slightly different in double precision format at runtime. These instances are rare enough that we can ignore them.

For Fugu-2019 or Fugu-CL however, our reproductions did not match in 6% and 19% of cases, whether we used the original C implementation or our own Python port. The Puffer team informed us of a use-after-free issue regarding the Transmission Control Protocol (TCP) info struct that was fixed in March 7th, 2022. Hence we retried this process for the logs pertaining to July 27th, 2022 and the error rate shrank to 0.53% and 0.64%. Unfortunately, a 0.5% error rate is still too high and even if we ignore that, limits us to RCT logs after March 7th. Therefore, we do not consider Fugu-2019 or Fugu-CL as candidates for left-out algorithms.

B.9 Training setup

We use Multi Layer Perceptrons (MLPs) as the NN structures for CausalSim models and the SLSim model. All implementations use the Pytorch [56] library. Table 3 is a comprehensive list of all hyperparameters used in training.

Appendix C Synthetic ABR

As explained in §6.3.1, we also 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.

C.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:

$$b_{t+1} = min(b_t - d_t, 0) + c \tag{20}$$

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, 10 seconds to mimic a live-stream ABR setting). To compute d_t, we model the transport as a TCP session with an Additive Increase - Multiplicative Decrease (AIMD)

Model	Hyperparameter	Value
SLSim (1 network), **CausalSim** (3 networks)	Hidden layers	(128, 128)
	Hidden layer Activation function	Rectified Linear Unit (ReLU)
	Output layer Activation function	Identity mapping
	Optimizer	Adam [40]
	Learning rate	0.001
	β_1	0.9
	β_2	0.999
	ε	10^{-8}
	Batch size	2^{17}
CausalSim	κ	{0.05, 0.1, 0.5, 1, 5, 10, 15, 20 ,25, 30, 40}
	Training iterations (num_train_it)	5000
	num_disc_it	10
	Loss function	Huber($\delta=0.2$)
	η (download time weight wrt buffer)	1
SLSim	Training iterations	10000
	Loss function	{Huber($\delta=0.2$), L1, MSE}
	η (download time weight wrt buffer)	{0.5, 1, 10}

Table 3: Training setup and hyperparameters for the real-world ABR experiment

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.

Performance Metric: We compare CausalSim predictions with ground truth counterfactual trajectories, via the Mean Squared Error (MSE) distance between the two time series:

$$MSE(\mathbf{p},\mathbf{q}) = ||\mathbf{p}-\mathbf{q}||_2^2 \tag{21}$$

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

C.1.1 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 [22].
- An ABR algorithm. We have a set of 9 policies to choose from, presented in Table 4.
- A network path, which is characterized by the latent network capacity and the path RTT.

We use random generative processes to generate 5000 network traces and RTTs. The RTT for a streaming session is sampled randomly, according to a uniform distribution:

$$rtt \sim Unif(10\,ms, 500\,ms)$$

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 [65], and Gaussian distributions can model throughputs in stationary segments of TCP flows [49].

Concretely, at the start of the trace, the following parameters

Policies	Hyperparameter	Value	Used as source	Used as left out
BBA	Cushion	5	✓	✓
	Reservoir	10		
BOLA-BASIC	V	0.71 (Computed using puffer formula)	✓	✓
	γ	0.22 (Computed using puffer formula)		
	Utility function	ln(chunk sizes) (As used in BOLA paper [63])		
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		
	Throughput estimate	Harmonic mean		
Rate-based	Lookback length	5	✓	✓
	Throughput estimate	Harmonic mean		
Optimistic Rate-based	Lookback length	5	✓	✓
	Throughput estimate	Max		
Pessimistic Rate-based	Lookback length	5	✓	✓
	Throughput estimate	Min		

Table 4: ABR algorithms used in the synthetic ABR experiments.

are randomly sampled:

$$\begin{aligned} v &\sim Unif(30, 100) \\ p &= 1/v \\ l, h &\sim Unif(0.5, 4.5) \\ s.t.\ &\frac{h-l}{h+l} > 0.3 \\ s_0 &\sim Unif(l, h) \\ c_\sigma &\sim Unif(0.05, 0.3) \end{aligned}$$

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

$$\begin{aligned} \lambda &= solve_{x\in\mathbb{R}^+}(1 - e^{x(h-s_{t-1})} - e^{x(s_{t-1}-l)} = 0) \\ s_t &= DoubleExp(s_{t-1}, \lambda) \end{aligned}$$

The point for this specific transition kernel is that small changes in network capacity should be more likely than drastic changes.

Finally, the network capacity c_t in each step is sampled from a Gaussian distribution, defined by these parameters:

$$c_t \sim Normal(s_t, s_t \cdot c_\sigma)$$

C.1.2 Training setup

Similar to the real-world ABR experiment, we use MLPs as the NN structures for CausalSim models and the SLSim model. We tune all the hyperparameters of both baselines as is done in the real-world ABR experiment (see §B.5 and §B.6). Table 5 comprehensively lists all hyperparameters used in training.

C.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

Model	Hyperparameter	Value
CausalSim (4 networks)	Hidden layers (SLSim)	(128, 128)
	Hidden layers (CausalSim: Extractor, Discriminator and $\mathcal{F}_{system}$)	(128, 128)
	Hidden layers (CausalSim: Action encoder)	(64, 64)
	Rank r	2
	Hidden layer Activation function	ReLU
	Output layer Activation function	Identity mapping
SLSim (1 network)	Optimizer	Adam [40]
	Learning rate	0.0001
	β_1	0.9
	β_2	0.999
	ε	10^{-8}
	Batch size	2^{13}
CausalSim	κ	{0.01, 0.1, 1, 10, 100}
	Training iterations (num_train_it)	20000
	num_disc_it	10
	Loss function	{MSE}
SLSim	Training iterations	20000
	Loss function	{Huber($\delta = 1.0$), L1, MSE}

Table 5: Training setup and hyperparameters for the synthetic ABR experiments.

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 §B.5. Figure 13a 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 13b gives a closer look at the CDF curves. We see CausalSim dominates at every scale.

Figure 13c 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 14, 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 froward in time. However, CausalSim maintains a MAPE of ($\sim$5.1%) which significantly lower than both ExpertSim's and SLSim's ($\sim$10%).

C.3 Learning ABR policies with CausalSim

We observed how CausalSim can be used to design an improved policy in §6.2, and verified this through deployment in the wild. We would like to take these experiments one step further and ask *can CausalSim be used to design learning-based policies, such as with Reinforcement Learning (RL)?*

Recent work has shown that RL algorithms can learn strong ABR policies by learning through interactions with the environment [50]. Could we use a CausalSim model to train high-performance ABR policies without direct environment interaction? As a first step, we decided to carry out an initial experiment in the synthetic ABR environment. We build a CausalSim model using traces from a "simulated RCT" on the synthetic environment.

Performance Metric. ABR algorithms are typically evaluated through QoE metrics [75]. Assuming the chosen bitrate at step t was q_t, the download time was d_t and the buffer was b_t, we use the following QoE definition:

$$QoE_t = q_t - |q_t - q_{t-1}| - \mu \cdot max(0, d_t - b_{t-1})$$

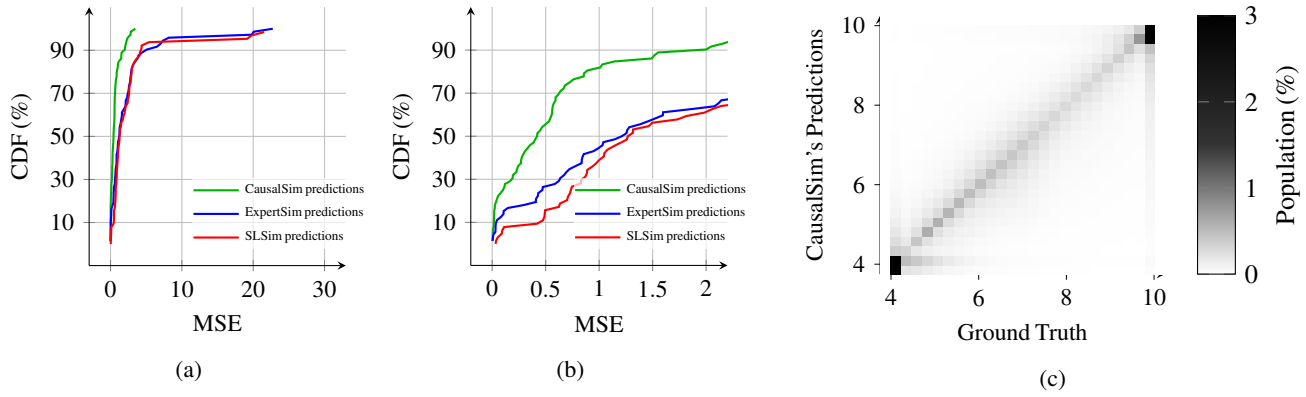


Figure 13: **(a)** Distribution of CausalSim, ExpertSim, and SLSim MSEs over all possible source left-out pairs. **(b)** The same figure with a smaller MSE range. In this magnified view, CausalSim clearly outperforms the baselines. **(c)** Two-dimensional histogram heatmap of CausalSim predictions vs. ground truth.

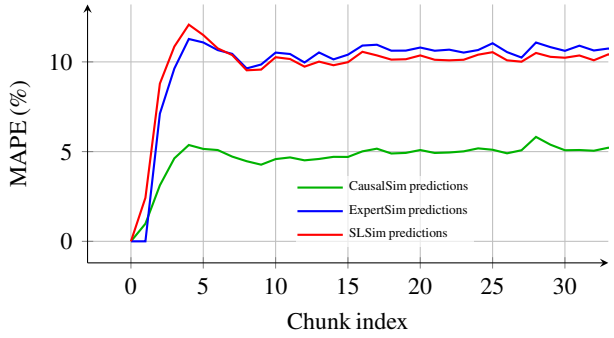


Figure 14: A time series plot of the Mean Absolute Percentage Error (MAPE) across all trajectories, for CausalSim, ExpertSim and SLSim predictions. Notice how errors accumulate in trajectory simulation.

This QoE metric captures three goals (in succession): 1) Stream in high quality, 2) Maintain a stable quality, 3) Avoid rebuffering. Better policies yield higher QoE values, where an ideal QoE is equal to the max bitrate.

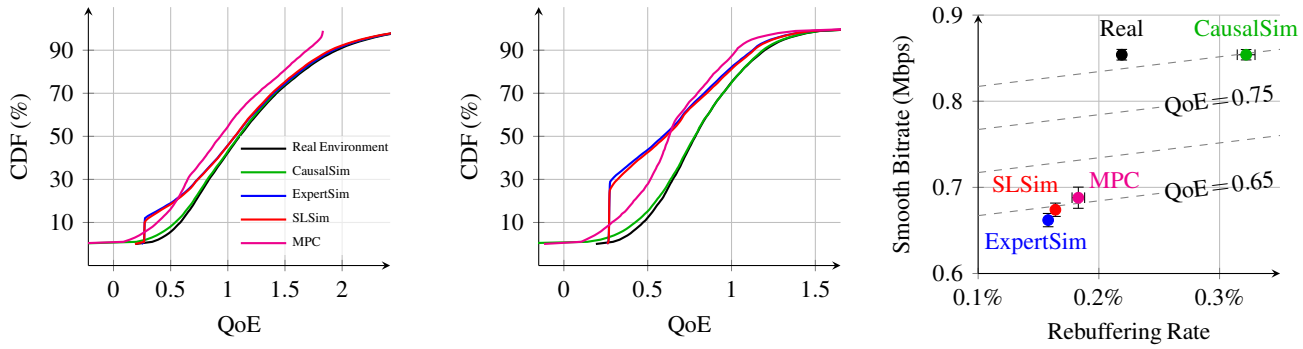
C.3.1 How to train policies via simulators?

To train the RL agent, we take a set of logged trajectories where the source policy was MPC and feed them to CausalSim. In each step, CausalSim will predict the next counterfactual observation and reward, and the RL agent will choose the next counterfactual action based on that observation. This process repeats until this simulated session is over, after which the counterfactual trajectory is used to train the RL agent. For the RL algorithm, we utilize the Advantage Actor Critic (A2C) method, a prominent on-policy algorithm, along with Generalized Advantage Estimation (GAE). Table 6 lists all hyperparameters for the RL training.

C.3.2 Does CausalSim train better policies?

Figure 15a plots the CDF of average session QoE that each policy attains. Here, *Real Environment* refers to training directly with the synthetic ABR environment, and CausalSim, ExpertSim and SLSim refer to policies trained by using each of these simulators. CausalSim trains policies nearly as well as training directly on the environment, while ExpertSim and SLSim fail to provide robust policies across all sessions. Figure 15b plots the CDFs for the high RTT (above 300 ms) clients, where the gap between CausalSim and the baseline simulators is even larger.

In this environment, chunk are downloaded according to the slow start model, where congestion control must ramp up its window size over several RTTs before the download rate can reach the available bandwidth. As a result, downloads of smaller chunks (with lower bitrates) incur a noticeable overhead, particularly on high-RTT paths. This overhead becomes less apparent as chosen bitrates become larger. Biased simulators such as SLSim and ExpertSim, which assume all actions lead to the same observed bandwidth, overestimate the achieved rate when counterfactual bitrates are smaller than factual ones (chosen by the source policy) and underestimate it when the counterfactual bitrates are larger. Since the source policy is conservative and tends to choose low bitrates, ExpertSim and SLSim find larger bitrates to be undesirable in the QoE trade-off. This can be seen in Figure 15c, which visualizes the 3 aspects of QoE in terms of the rebuffering rate and the smoothed birate, i.e the chosen bitrates with the smoothnes penalty. Notice how policies trained on the real environment and CausalSim utilize the network by 200 kbps more than other policies. The extra rebuffering that CausalSim incurs is negligible compared to the extra bitrate: 5.9 seconds every hour.



(a) Full population

(b) High RTT clients

(c) QoE breakdown in High RTT clients

Figure 15: CausalSim trained policies perform well, only marginally behind training on the real environment. Distribution of Quality of Experience (QoE) in policies trained with the real environment, CausalSim, ExpertSim, and the MPC policy. CausalSim does not underestimate bandwidth in high RTT clients and trains policies that strike the best balance in QoE goals.

Group	Hyperparameter	Value
Neural Network	Hidden layers	(32, 32)
	Hidden layer activation function	ReLU
	Output layer activation function	A2C actor: Softmax
		A2C critic: Identity mapping
	Optimizer	Adam [40]
	Learning rate	0.001
	β_1	0.9
	β_2	0.999
	ϵ	10^{-8}
	Weight decay	10^{-4}
A2C training	Episode lengths	490
	Epochs to convergence (T_c)	8000 (3920000 samples)
	Random seeds	4
	γ	0.96
	Entropy schedule	0.1 to 0 in 5000 epochs
	λ (for GAE)	0.95
Environment	Chunk length c	4
	Number of actions (bitrates)	6

Table 6: Training setup and hyperparameters for learning RL policies in the synthetic ABR environment.

C.4 Low-rank structure

As discussed in §4.1, we can formulate the counterfactual estimation problem in the context of matrix completion. For each time step, we know the chosen bitrate (action) and the achieved throughput (trace). We also know the trace is computed using a latent factor and the action. Suppose the latent factor is the network bottleneck capacity c_t[18]. $\mathcal{F}_{\text{trace}}$ describes how the achieved throughput (the trace) relates to this latent factor. Intuitively, this should be a close-to-linear function, $m_t \approx c_t$. But it's not exactly linear; for example, congestion control may under-utilize the network capacity for

[18]There may be other latent factors but bottleneck capacity is likely to have the strongest influence on the achieved throughput.

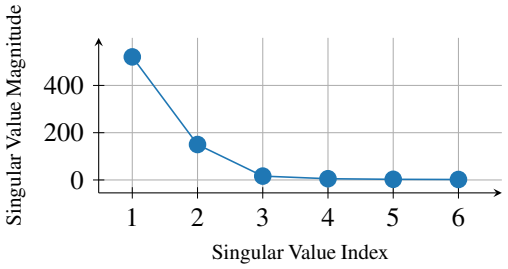


Figure 16: Singular values of matrix M in synthetic ABR suggest that M is approximately rank 2.

small transfers on high-RTT paths.

We form a matrix M, where the rows denote actions $a_t \in [A]$ and the columns denote the latent factors u_t^i for each trajectory. The 'factual' data we have are single observed trace values in each column, i.e for each step and each latent, we have observed the trace from a single action. To estimate counterfactuals, we must complete the matrix. We have no way of knowing the true $\mathcal{F}_{\text{trace}}$ in the Puffer dataset. But to get a sense for what it might look like and whether it's plausible that M is low rank, we can investigate this in the synthetic ABR environment instead.

For the TCP slow start model this environment uses, $\mathcal{F}_{\text{trace}}$ takes the following form:

$$\text{Let } \hat{RTT} := \frac{RTT}{\ln(2)} \tag{22}$$

$$m_t = \begin{cases} \dfrac{c_t}{1+\frac{\hat{RTT}\cdot(\ln(c_t/\dot{c})-c_t+\dot{c})}{s_t}} & \text{if } s_t \geq \hat{RTT}.(c_t-\dot{c}) \\ \dfrac{s_t}{\hat{RTT}\cdot\ln(\frac{s_t}{\hat{RTT}\cdot\dot{c}}+1)} & \text{otherwise} \end{cases} \tag{23}$$

where s_t is the chunk size (which itself is determined by the bitrate chosen by ABR) and $\dot{c}$ is the starting download rate in the slow start algorithm (in our case, equal to 2 MTUs). We use this model to generate a version of M with $A = 6$ actions and $U = 49000$ latent network conditions. We compute the singular value decomposition with the 6 singular values represented in non-increasing order ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_6$). The total "energy" of matrix is given by sum of squares of these singular values. It turns out that $\frac{\sigma_1^2+\sigma_2^2}{\text{total energy}}$ is more than 0.999. This suggests that most of the matrix is captured by its rank-2 approximation, as depicted in Figure 16. In other words, M is approximately low (=2) rank.

Appendix D Load Balancing

D.1 Does CausalSim Faithfully Infer Latent States?

We test the claim that estimating the exogenous latent state and using it to predict the next state was indeed the key to producing accurate counterfactual predictions, as the architecture of CausalSim suggests. 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. We find that the estimated latent states and the job sizes are highly correlated, as illustrated in Figure 17, with a PCC of 0.994. This demonstrates that CausalSim can learn faithful representations of true latent states.

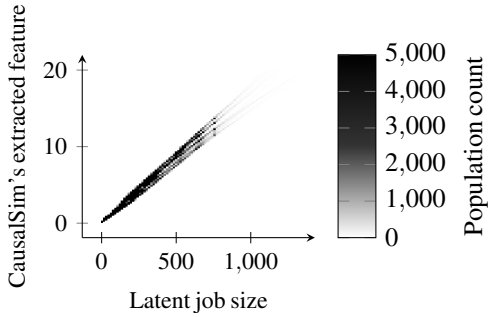


Figure 17: Two-dimensional histogram heatmap of CausalSim extracted latent state vs. latent job sizes.

D.2 Data & Algorithms

To simulate the load balancing problem described in §6.4.1, we need to set the server processing rates $\{r_i\}_{i=1}^N$, and arriving job sizes S_k. Server rates are generated randomly, as follows:

$$r_i = e^{u_i} \tag{24}$$

$$\text{where } u_i \sim Unif(-\ln(5), \ln(5)) \tag{25}$$

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

$$S_k \sim Normal(\mu_k, \sigma_k)$$

where μ_k and σ_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 values are drawn from random distributions, both at the start of a trajectory and when a change occurs, in the following manner:

If $k=0$ (start of trace) or, mean and variance must change:

$$\mu_k \sim Pareto(\alpha=1,\ L=10^1,\ H=10^{2.5}) \tag{26}$$

$$\sigma_k \sim Unif(0,\ 0.5\mu_k) \tag{27}$$

Else:

$$\mu_k = \mu_{k-1} \tag{28}$$

$$\sigma_k = \sigma_{k-1} \tag{29}$$

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 a set of 16 policies, described in Table 7.

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	✓	✓

Table 7: Scheduling policies used in the load balancing experiment.

D.3 Training setup

As before, we use MLPs as the NN structures for CausalSim models and the SLSim model and Table 8 is a comprehensive list of all hyperparameters used in training. We tune the parameter κ for CausalSim and the loss function in SLSim in a similar fashion to what is described in §B.5 and §B.6. Note that, as mentioned in §6.4.1, we assume access to $\mathcal{F}_{\text{system}}$ and focus on the more challenging task of estimating the trace quantities, for both CausalSim and SLSim. Therefore, in training, there are no observations and hence $\mathcal{L}_{\text{total}}$ consist of two terms: the squared loss of the trace quantities and the discriminator loss.

Appendix E Causal Inference Related Work

Identifying causal relationships from observational data is a critical problem in many domains [30], including medicine [55], epidemiology [59], economics [36], and education [23]. Indeed, identifying causal structure and answering causal inference queries is an emerging theme in different machine learning tasks recently, including computer vision [74, 76], reinforcement learning [6, 24], fairness [27], and time-series analysis [7] to name a few. One important aspect about causal inference is its ability to answer counterfactual queries. For such queries, many methods were developed; where some approaches are motivated by Pearl's structural causal model [57], and by Rubin's potential outcome framework [61]. We refer the interested reader to recent surveys such as [30] and references there in for an overview of recent advances in our ability to infer causal relationships from observational data.

Another related line of work within this literature 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 [4, 5, 9–12]. However, these approaches assume a static set of intervention and do not apply to our setting.

Model	Hyperparameter	Value
CausalSim (3 networks)	Hidden layers (SLSim)	(128, 128)
	Hidden layers (CausalSim: Extractor, Discriminator)	(128, 128)
	Hidden layers (CausalSim: Action encoder)	No hidden layers
	Rank r	1
	Hidden layer Activation function	ReLU
	Output layer Activation function	Identity mapping
	Optimizer	Adam [40]
SLSim (1 network)	Learning rate	0.0001
	β_1	0.9
	β_2	0.999
	ε	10^{-8}
	Batch size	2^{13}
CausalSim	κ	$\{0.01, 0.1, 1, 10, 100\}$
	Training iterations (num_train_it)	10000
	num_disc_it	10
SLSim	Training iterations	10000
	Loss function	Huber, L1, MSE

Table 8: Training setup and hyperparameters for the load balancing experiment.